The Complex(ity) Landscape of Checking Infinite Descent

LIRON COHEN, Ben-Gurion University of the Negev, Israel
ADHAM JABARIN, Ben-Gurion University of the Negev, Israel
ANDREI POPESCU, University of Sheffield, UK
REUBEN N. S. ROWE, Royal Holloway University of London, UK

Cyclic proof systems, in which induction is managed implicitly, are a promising approach to automatic verification. The soundness of cyclic proof graphs is ensured by checking them against a trace-based Infinite Descent property. Although the problem of checking Infinite Descent is known to be PSPACE-complete, this leaves much room for variation in practice. Indeed, a number of different approaches are employed across the various cyclic proof systems described in the literature. In this paper, we study criteria for Infinite Descent in an abstract, logic-independent setting. We look at criteria based on Büchi automata encodings and relational abstractions, and determine their parameterized time complexities in terms of natural dimensions of cyclic proofs: the numbers of vertices of the proof-tree graphs, and the *vertex width*—an upper bound on the number of components (e.g., formulas) of a sequent that can be simultaneously tracked for descent. We identify novel algorithms that improve upon the parameterised complexity of the existing algorithms. We implement the studied criteria and compare their performance on various benchmarks.

CCS Concepts: • **Theory of computation** → **Automated reasoning**; **Logic and verification**; **Automata over infinite objects**; *Complexity theory and logic*; *Fixed parameter tractability*.

Additional Key Words and Phrases: cyclic proof, infinite descent, size-change termination, complexity, algorithms, Büchi automata

ACM Reference Format:
Liron Cohen, Adham Jabarin, Andrei Popescu, and Reuben N. S. Rowe. 2024. The Complex(ity) Landscape of Checking Infinite Descent. *Proc. ACM Program. Lang.* 8, POPL, Article 46 (January 2024), 33 pages. https://doi.org/10.1145/3632888

1 INTRODUCTION

Non-wellfounded, or cyclic, reasoning [Brotherston and Simpson 2011; Cohen and Rowe 2020; Das 2020; Dax et al. 2006; Doumane 2017; Santocanale 2002; Sprenger and Dam 2002] has gained significant attention in the field of program verification. Indeed, various work on verifying programs and inductive properties, especially in the context of Separation Logic, has been based on cyclic reasoning techniques [Brotherston et al. 2008; Brotherston and Gorogiannis 2014; Rowe and Brotherston 2017; Ta et al. 2016, 2018; Tellez and Brotherston 2020], leading to the development of the Cyclist [Brotherston et al. 2012], Songbird [Cheng et al. 2016], and Inductor [Serban and Iosif 2018] provers. These techniques have also been employed in the Cypress program synthesis tool [Itzhaky et al. 2021], in the CycleQ prover for equational reasoning about functional programs [Jones et al. 2022], and for automatically verifying termination of functional programs [Lepigre and Raffalli 2019].

Authors' addresses: Liron Cohen, Ben-Gurion University of the Negev, Beersheba, Israel, cliron@cs.bgu.ac.il; Adham Jabarin, Ben-Gurion University of the Negev, Beersheba, Israel, adhamj@post.bgu.ac.il; Andrei Popescu, University of Sheffield, Sheffield, UK, a.popescu@sheffield.ac.uk; Reuben N. S. Rowe, Royal Holloway University of London, London, UK, reuben.rowe@rhul.ac.uk.

ACM 2475-1421/2024/1-ART46
https://doi.org/10.1145/3632888

Cyclic reasoning provides a means for *implicit* induction, as opposed to the standard technique of *explicit* induction. Instead of cluttering the proof with inductive assumptions, the implicit approach exploits the presence of cycles to encode inductive invariants. For example, while attempting to prove $P(x)$, one repeatedly decomposes the goal into subgoals that are either provable or reducible to $P(x)$. In the latter case, a cycle in the reasoning is employed, but some form of 'progress' during decomposition is needed to avoid inconsistency. This notion of 'progress' is formalised as an Infinite Descent condition (cf. Fermat's *Descente Infinie*). Another form of the Infinite Descent condition used in the context of program termination analysis is the Size-Change Principle [Lee et al. 2001]. There, instead of guaranteeing that a cyclic proof is valid, the principle guarantees that a recursive program exhibiting a given function call pattern terminates. The automatic verification of the Infinite Descent condition is therefore a crucial component when deploying cyclic reasoning in practice.

Several methods for deciding Infinite Descent have been proposed, in the context of various concrete cyclic proof systems. These can be categorised into two groups: those based on ω-automata, e.g. [Brotherston 2006; Dax et al. 2006; Simpson 2017], and those involving relation-based criteria, e.g. [Hazard and Kuperberg 2022; Jones et al. 2022]. The characterisation in terms of ω-automata captures Infinite Descent by checking an inclusion between two ω-regular languages: one denoting all such infinite chains, or paths, and one denoting all 'infinitely progressing' paths, respectively. The relation-based criteria work by checking Ramsey-theoretic properties.

Despite the diversity of approaches, there has been comparatively little focus on the practicality of these methods, particularly in the context of automated theorem proving. Deciding Infinite Descent has been shown to be PSPACE-complete [Lee et al. 2001; Nollet et al. 2019]. However, this characterisation does not really lead to a good understanding of how these algorithms compare to one another in practice. Moreover, the various methods for deciding Infinite Descent have been discussed within different contexts and communities. As cyclic reasoning becomes more widely used and integrated into provers, it is increasingly important to bring these methods under the same roof and to understand their complexity, as well as their relative strengths and limitations.

Our interest is in the time complexity of algorithms for deciding Infinite Descent. In the automaton-based algorithms, the complexity is dominated by the complementation of an automaton of size k polynomial in the input proof, an operation of complexity $O(2^{k \cdot \log k})$. For the relation-based algorithms, although a notion of subsumption between the relations allows an optimisation in which some relations can be disregarded [Ben-Amram and Lee 2007; Fogarty and Vardi 2009], in the worst case it is still necessary to generate exponentially many.

However, in the context of cyclic proofs, it is natural to further decompose the input along two axes. The information relevant to deciding Infinite Descent comprises (1) the vertices in the proof graph and (2) the ways a trace can interact with the logical judgments attached to each vertex. This leads us to a fine-grained *parameterised* complexity analysis, which better differentiates the relative (worst-case) performance of the different algorithms. We propose a new automaton-based construction that is polynomial in the number of vertices, and exponential only in the vertex width (i.e. the number of possible positions within a vertex that a trace may visit). This contrasts with the automaton-based construction commonly described, which is exponential in both parameters. We further note that the relation-based approach already displays a separation in the complexity of these parameters: polynomial in the number of vertices, exponential in the vertex width. We propose a novel variation of the relation-based method that reduces the exponential degree in the vertex width.

Finally, since we are also interested in practical performance, we implemented our new algorithms, both relation-based and automaton-based, within the theorem prover Cyclist. We compared the performance of these methods to its existing automaton-based algorithm across a number of benchmarks, and found that our novel relation-based algorithm provided clear improvements.

In this paper, we make the following contributions.

- We carry out the first comparative analysis of the parameterised time complexity of methods for deciding Infinite Descent in cyclic proof systems.
- We propose and prove correct a novel relation-based optimisation.
- We describe, abstractly, a new automaton-theoretic approach, and relate it to the standard approach from the literature.
- We implement the aforementioned algorithms within the Cyclist theorem prover, and compare their performance in practice.

Outline. Section 2 provides background on cyclic proofs and Infinite Descent, and discusses the abstraction we use for our comparative analysis. Section 3 gives an overview of the complexity landscape of Infinite Descent checking algorithms. Sections 4 and 5 discuss the automaton-based and relation-based algorithms, respectively. Section 6 describes our implementation and experimental evaluation. Section 7 discusses related work and Section 8 concludes.

2 CYCLIC PROOFS AND THE INFINITE DESCENT PROPERTY

This section recalls cyclic proofs and their Infinite Descent property. It also describes a general setting that abstracts away logical formalism details that are irrelevant to Infinite Descent.

2.1 Background on Cyclic Proofs

The starting point for the theory of cyclic proof systems are standard (finitary) proof systems, given by rules for inferring sequents (or whatever type of statements that can be made in the given logic). More detailed background may be found in an appendix included as supplementary material.

Definition 2.1. A *proof system* is a tuple $(\mathsf{Seq}, \mathsf{Rule}, \mathsf{Hyps}, \mathsf{conc})$ where: Seq is a set of *sequents*; Rule is a set of *rules*; and $\mathsf{Hyps} : \mathsf{Rule} \to \mathscr{P}(\mathsf{Seq})$ and $\mathsf{conc} : \mathsf{Rule} \to \mathsf{Seq}$ are functions associating to each rule rl its finite set of hypotheses $\mathsf{Hyps}(rl)$ and its conclusion $\mathsf{conc}(rl)$. We write $\dfrac{seq_1 \dots seq_n}{seq}\ rl$ for a rule rl with hypotheses $seq_1, \dots, seq_n$ and conclusion seq.

Example 2.2 (The proof system $\mathsf{LKID}(\mathsf{BasicNat})$*).* A FOLID (FOL with Inductive Definitions) language [Brotherston 2006; Brotherston and Simpson 2011] is a triple $(\mathsf{Fsym}, \mathsf{Psym}, \mathsf{IPsym})$ where $(\mathsf{Fsym}, \mathsf{Psym})$ is a first-order language with a set of function symbols Fsym and a set of predicates Psym, and $\mathsf{IPsym} \subseteq \mathsf{Psym}$ is a distinguished subsets of predicates that are deemed *inductive*. We consider the FOLID theory BasicNat having the language $\mathsf{Fsym} = \{0, \mathrm{s}\}$, $\mathsf{IPsym} = \mathsf{Psym} = \{N, O, E\}$, and the defining clauses for N, O and E as follows:

$$N(0) \text{ and } N(y) \longrightarrow N(\mathrm{s}(y)) \quad ; \quad E(0) \text{ and } O(y) \longrightarrow E(\mathrm{s}(y)) \quad ; \quad E(y) \longrightarrow O(\mathrm{s}(y)).$$

The proof system for the FOLID theory BasicNat, written $\mathsf{LKID}(\mathsf{BasicNat})$, extends Gentzen's proof system LK [Gentzen 1935] with two types of rules: for each n-ary inductive predicate $p \in \mathsf{IPsym}$, a case splitting rule $\textsc{Split}_p$ and, for each formula θ in the defining clauses for p, an introduction rule $\textsc{Intr}_{p,\theta}$. The $\mathsf{LKID}(\mathsf{BasicNat})$ introduction and splitting rules associated to N, O and E are the following, where we write, for example, $\textsc{Intr}_{N,0}$ instead of $\textsc{Intr}_{N,N(0)}$ and $\textsc{Intr}_{N,\mathrm{s}}$ instead of $\textsc{Intr}_{N,N(x)\longrightarrow N(\mathrm{s}(x))}$:

$$\frac{}{\Gamma \vdash \Delta, N(0)}\ \textsc{Intr}_{N,0} \qquad \frac{\Gamma \vdash \Delta, N(y)}{\Gamma \vdash \Delta, N(\mathrm{s}(y))}\ \textsc{Intr}_{N,\mathrm{s}}$$

$$\frac{\Gamma[0/x] \vdash \Delta[0/x] \quad \Gamma[\mathrm{s}(y)/x], N(y) \vdash \Delta[\mathrm{s}(y)/x]}{\Gamma, N(x) \vdash \Delta}\ \textsc{Split}_N\ (y \text{ fresh})$$

$$\dfrac{\dfrac{}{2\colon\ \vdash E(0), O(0)}\,\textsc{Intr}_{E,0} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{6\colon N(x) \vdash O(x), E(x)}\,\textsc{Subst}}{5\colon N(y) \vdash O(y), E(y)}\,\textsc{Intr}_{O,\mathsf{s}}}{4\colon N(y) \vdash O(y), O(\mathsf{s}(y))}\,\textsc{Intr}_{E,\mathsf{s}}}{3\colon N(y) \vdash E(\mathsf{s}(y)), O(\mathsf{s}(y))}\,\textsc{Split}_N}{\dfrac{1\colon N(x) \vdash E(x), O(x)}{0\colon N(x) \vdash E(x) \vee O(x)}\,\vee\mathrm{R}}$$

Fig. 1. An open proof tree, turned into a cyclic proof (a variant of Ex. 5.2.1 from Brotherston [2006]).

$$\frac{}{\Gamma\ \vdash \Delta, E(0)}\,\textsc{Intr}_{E,0} \qquad \frac{\Gamma \vdash \Delta, O(y)}{\Gamma \vdash \Delta, E(\mathsf{s}(y))}\,\textsc{Intr}_{E,\mathsf{s}}$$

$$\frac{\Gamma[0/x] \vdash \Delta[0/x] \quad \Gamma[\mathsf{s}(y)/x], O(y) \vdash \Delta[\mathsf{s}(y)/x]}{\Gamma, E(x) \vdash \Delta}\,\textsc{Split}_E\ (y \text{ fresh})$$

$$\frac{\Gamma \vdash \Delta, E(y)}{\Gamma \vdash \Delta, O(\mathsf{s}(y))}\,\textsc{Intr}_{O,\mathsf{s}} \qquad \frac{\Gamma[\mathsf{s}(y)/x], E(y) \vdash \Delta[\mathsf{s}(y)/x]}{\Gamma, O(x) \vdash \Delta}\,\textsc{Split}_O\ (y \text{ fresh})$$

Definition 2.3. Given a proof system $(\mathsf{Seq}, \mathsf{Rule}, \mathsf{Hyps}, \mathsf{conc})$, an *open derivation tree* is a tuple $(\mathsf{V}, \mathsf{E}, \mathsf{seqOf}, \mathsf{Bud})$ where (V, E) is a finite tree, $\mathsf{seqOf} : \mathsf{V} \to \mathsf{Seq}$ is a labeling of the vertices with sequents, and $\mathsf{Bud} \subseteq \mathsf{V}$ is a subset of vertices called *buds* (unexpanded leaves), such that every non-bud vertex $v \in \mathsf{V} \setminus \mathsf{Bud}$ corresponds to a rule application and the buds have no outgoing edges.

Open derivation trees model possibly incomplete proofs. A proof development starts with a goal sequent *seq* and repeatedly applies rules backwards. At any moment, we have an open derivation tree whose root is labeled with *seq*, and whose buds are labeled with the pending goals.

Example 2.4. Fig. 1 shows an open derivation tree for the sequent $N(x) \vdash E(x) \vee O(x)$ in the proof system LKID(BasicNat) from Example 2.2. We have $\mathsf{V} = \{0, 1, 2, 3, 4, 5, 6\}$, $\mathsf{E} = \{(0, 1), (1, 2), (1, 3), (3, 4), (4, 5), (5, 6)\}$, $\mathsf{seqOf}(0) = (N(x) \vdash E(x) \vee O(x))$, $\mathsf{seqOf}(1) = (N(x) \vdash E(x), O(x))$, etc. This derivation is an incomplete proof because it has a bud, namely $\mathsf{Bud} = \{6\}$.

In the Fig. 1 derivation tree, the sequents labeling vertices 1 and 6 are identical (assuming sequents comprise sets, not lists), and so, intuitively, we can "backlink" vertex 6 to vertex 1 and create a cycle, thus "closing" the proof tree within a cyclic proof. Cyclic proofs are of course not necessarily sound: The sequent $N(x) \vdash E(x), O(x)$ (labeling vertex 1) is being justified using itself as an assumption (labeling vertex 6). However, we have a way to detect descent along this cycle: In between these two occurrences of the same sequent, there was a $(\textsc{Split}_N)$ rule applied (backwards) to the predicate instance $N(x)$ which turned it into the smaller $N(y)$, afterwards renamed back into $N(x)$ at vertex 6—where "smaller" means "generated sooner by the inductive semantics". Thus, the occurrence of $N(x)$ at vertex 6 is, in a precise sense, smaller than that at vertex 1. The cycle is not "flat" but contains a "downward slope" from 1 to 6—meaning the "position" of one of the formulas in the sequent, $N(x)$, has decreased between 1 and 6, validating the cyclic reasoning. The above idea is captured by the Infinite Descent property. It states that every infinite path built from edges and backlinks has, starting from some point, a trace of positions that: (1) is "descending" in the sense that any two consecutive positions are related by a "maintain" ($\rightsquigarrow$) or "decrease" ($\leadsto$) slope; and (2) always eventually the trace is guaranteed to be "strictly descending", in that it contains two positions related by a "decrease" slope.

Our running example satisfies Infinite Descent: Its only infinite path is $0{\cdot}(1{\cdot}3{\cdot}4{\cdot}5{\cdot}6)^\omega$, which starting from its second position has the trace of positions $(N(x) \leadsto N(y) \rightsquigarrow N(y) \rightsquigarrow N(y) \rightsquigarrow N(x))^\omega$.

To make this notion of progress formal, we augment proof systems with the notions of a position, and of a slope relating two positions. For this, we fix a set Pos ranged over by p, p', etc., of items that we call *positions*. Note that this set is associated to an entire proof system, not to a particular proof tree. In what follows we will give an example for a concrete choice of this set. We let S, ranged over by s, s' etc., be the two-element set $\{\rightsquigarrow, \leadsto\}$. We think of $\rightsquigarrow$ and $\leadsto$ as representing the *slope* of each individual step along a trace through a proof. Given $K, K' \subseteq \mathsf{Pos}$, a *sloped relation* between K and K' is a ternary relation $R \subseteq K \times K' \times \mathsf{S}$ that is single-valued (i.e., a partial function), in that if $R(p, p', s_1)$ and $R(p, p', s_2)$ then $s_1 = s_2$. Thus, a sloped relation indicates the slope of the segment of a trace between two positions: non-increasing if $R\,(p, p', \rightsquigarrow)$ holds, and decreasing if $R\,(p, p', \leadsto)$ holds.

Definition 2.5. A *sloped proof system* is a tuple $\mathsf{SPS} = (\mathsf{Seq}, \mathsf{Rule}, \mathsf{Hyps}, \mathsf{conc}, \mathsf{Pos}, \mathsf{Ps}, \mathsf{R})$ where:

- $(\mathsf{Seq}, \mathsf{Rule}, \mathsf{Hyps}, \mathsf{conc})$ is a proof system;
- Pos, ranged over by p, p', etc., is a finite set of items called *positions*;
- $\mathsf{Ps} : \mathsf{Seq} \to \mathscr{P}(\mathsf{Pos})$ is a function associating to every sequent a set of positions;
- R is an indexed family $(\mathsf{R}_{rl,seq})_{rl \in \mathsf{Rule}, seq \in \mathsf{Hyps}(rl)}$, where $\mathsf{R}_{rl,seq} \subseteq \mathsf{Ps}(\mathsf{conc}(rl)) \times \mathsf{Ps}(seq) \times \mathsf{S}$ i.e., $\mathsf{R}_{rl,seq}$ is a sloped relation between the positions of $\mathsf{conc}(rl)$ and those of seq.

Example 2.6. The sloped proof system $\mathsf{LKID}^\omega(\mathsf{BasicNat})$ extends $\mathsf{LKID}(\mathsf{BasicNat})$ by defining the positions to be all instances of the three inductive predicates: $\mathsf{Pos} = \{N(t) \mid t \in \mathsf{Term}\} \cup \{E(t) \mid t \in \mathsf{Term}\} \cup \{O(t) \mid t \in \mathsf{Term}\}$ and, for each seq, $\mathsf{Ps}(seq)$ gives all positions in seq. The only decreases occur along the backwards application of the split rule. For example, if $seq = (\Gamma,\ N(x) \vdash \Delta)$ is the conclusion of $\textsc{Split}_N$ and $seq' = \Gamma[\mathsf{s}(y)/x],\ N(y) \vdash \Delta[\mathsf{s}(y)/x]$ is its second hypothesis, then $\mathsf{R}_{\textsc{Split}_N, seq'}(N(x), N(y)) = \leadsto$. And similarly for $\textsc{Split}_E$ and $\textsc{Split}_O$.

Definition 2.7. A *cyclic preproof* in a sloped proof system SPS is a tuple of the form $(\mathsf{V}, \mathsf{E}, \mathsf{seqOf}, \mathsf{Bud}, \mathsf{backlink})$ where $(\mathsf{V}, \mathsf{E}, \mathsf{seqOf}, \mathsf{Bud})$ is a derivation tree in the underlying proof system and $\mathsf{backlink} : \mathsf{Bud} \to \mathsf{V} \setminus \mathsf{Bud}$ is a function that maps each bud to a non-bud vertex labeled with the same sequent: $\forall v \in \mathsf{Bud}.\ \mathsf{seqOf}(\mathsf{backlink}(v)) = \mathsf{seqOf}(v)$.

Example 2.8. The derivation tree in Fig. 1 becomes a cyclic preproof in the sloped proof system $\mathsf{LKID}^\omega(\mathsf{BasicNat})$ by defining $\mathsf{Bud} = \{6\}$ and $\mathsf{backlink} : \mathsf{Bud} \to \mathsf{V} \setminus \{6\}$ by $\mathsf{backlink}(6) = 1$.

Example 2.9 (The proof system $\mathsf{LKID}(\mathsf{Hydra})$*).* We consider a second running example, namely the encoding of the 2-Hydra problem due to Berardi and Tatsuta [2017]. Hydra is the FOLID theory having the language $\mathsf{Fsym} = \{0, \mathsf{s}\}$, $\mathsf{IPsym} = \mathsf{Psym} = \{N, H\}$ (with H binary), the defining clauses for N as before (the same as for $\mathsf{LKID}(\mathsf{BasicNat})$), and the defining clauses for H as follows:

$$(1)\ H(0,0)\ \ ;\ \ (2)\ H(\mathsf{s}(0),0)\ \ ;\ \ (3)\ H(x,\mathsf{s}(0))\ \ ;\ \ (4)\ H(x,y) \longrightarrow H(\mathsf{s}(x),\mathsf{s}(\mathsf{s}(y)))$$
$$(5)\ H(\mathsf{s}(y),y) \longrightarrow H(0,\mathsf{s}(\mathsf{s}(y)))\ \ ;\ \ (6)\ H(\mathsf{s}(x),x) \longrightarrow H(\mathsf{s}(\mathsf{s}(x)),0)$$

We will refer to the introduction rules associated to the above 6 clauses by $\textsc{Intr}_{H,i}$ for i ranging from 1 to 6. The sloped proof system $\mathsf{LKID}^\omega(\mathsf{Hydra})$ extends $\mathsf{LKID}(\mathsf{Hydra})$ in the same way in which $\mathsf{LKID}^\omega(\mathsf{BasicNat})$ extends $\mathsf{LKID}(\mathsf{BasicNat})$ (Example 2.6). Fig. 2 shows a cyclic preproof in $\mathsf{LKID}^\omega(\mathsf{Hydra})$ for the fact that the hydra predicate $H(x, y)$ holds for all natural numbers x and y.

2.2 Infinite Descent for Sloped Graphs

Next, we note that Infinite Descent can be expressed and studied abstractly, without actual references to proof systems or proof trees. All we need to remember about a cyclic proof is its associated

$$\dfrac{\dfrac{\Pi_1}{1\colon N(x) \vdash H(x,0)} \quad \dfrac{}{9\colon N(x) \vdash H(x,\mathsf{s}(0))}\,\textsc{Intr}_{H,3} \quad \dfrac{\Pi_2}{10\colon N(x), N(\mathsf{s}(y'')), N(y'') \vdash H(x,\mathsf{s}(\mathsf{s}(y'')))}}{8\colon N(x), N(y') \vdash H(x,\mathsf{s}(y'))}\,\textsc{Split}_N}{0\colon N(x), N(y) \vdash H(x,y)}\,\textsc{Split}_N$$

(a) The root of the proof

$$\dfrac{\dfrac{}{2\colon \vdash H(0,0)}\,\textsc{Intr}_{H,1} \quad \dfrac{\dfrac{}{4\colon \vdash H(\mathsf{s}(0),0)}\,\textsc{Intr}_{H,2} \quad \dfrac{\dfrac{7\colon N(x), N(y) \vdash H(x,y)}{6\colon N(\mathsf{s}(x'')), N(x'') \vdash H(\mathsf{s}(x''),0)}\,\textsc{Subst}}{5\colon N(\mathsf{s}(x'')), N(x'') \vdash H(\mathsf{s}(\mathsf{s}(x'')),0)}\,\textsc{Intr}_{H,6}}{3\colon N(x') \vdash H(\mathsf{s}(x'),0)}\,\textsc{Split}_N}{1\colon N(x) \vdash H(x,0)}\,\textsc{Split}_N$$

(b) The subderivation Π_1

$$\dfrac{\dfrac{\dfrac{13\colon N(x), N(y) \vdash H(x,y)}{12\colon N(\mathsf{s}(y'')), N(y'') \vdash H(\mathsf{s}(y''),0)}\,\textsc{Subst}}{11\colon N(\mathsf{s}(y'')), N(y'') \vdash H(0,\mathsf{s}(\mathsf{s}(y'')))}\,\textsc{Intr}_{H,5} \quad \dfrac{\dfrac{\dfrac{17\colon N(x), N(y) \vdash H(x,y)}{16\colon N(x'), N(y'') \vdash H(x',y'')}\,\textsc{Subst}}{15\colon N(x'), N(\mathsf{s}(y'')), N(y'') \vdash H(x',y'')}\,\textsc{WeakL}}{14\colon N(x'), N(\mathsf{s}(y'')), N(y'') \vdash H(\mathsf{s}(x'),\mathsf{s}(\mathsf{s}(y'')))}\,\textsc{Intr}_{H,4}}{10\colon N(x), N(\mathsf{s}(y'')), N(y'') \vdash H(x,\mathsf{s}(\mathsf{s}(y'')))}\,\textsc{Split}_N$$

(c) The subderivation Π_2

Fig. 2. A cyclic preproof in LKID$^\omega$(Hydra) of the totality of the 2-Hydra predicate.

graph and the "sloped relations" between the positions in the vertices along each edge in the graph. This information can be packed into the following concise structure.

Definition 2.10. A *sloped graph* $\mathsf{SG} = (\mathsf{V}, \mathsf{E}, \mathsf{Ps}, \mathsf{R})$ is a directed graph (V, E) with the following additional components:

- $\mathsf{Ps} : \mathsf{V} \to \mathscr{P}(\mathsf{Pos})$ is a mapping from vertices to sets of positions, and
- R is an edge-indexed family of sloped relations $(\mathsf{R}_{v,v'})_{(v,v')\in\mathsf{E}}$ where $\mathsf{R}_{v,v'}$ is a sloped relation between $\mathsf{Ps}(v)$ and $\mathsf{Ps}(v')$.

We call the value $\mathsf{Max}\,\{|\mathsf{Ps}(v)| \mid v \in \mathsf{V}\}$ the *vertex width* of SG.

Thus, a sloped graph is a directed graph together with an indication of the possible positions on its vertices, and slope changes along its edges. It abstracts away from sequents and rules.

Example 2.11. The sloped graph corresponding to the cyclic preproof of Example 2.8 has the following components:

- $\mathsf{V} = \{0,1,2,3,4,5\}$ (all vertices of the proof tree except for the bud);
- $\mathsf{E} = \{(0,1),(1,2),(1,3),(3,4),(4,5),(5,1)\}$ (all edges of the tree except for that to the bud, $(5,6)$, which is replaced with an edge via the backlink, $(5,1)$);
- $\mathsf{Ps}(0) = \mathsf{Ps}(1) = \{N(x)\}$, $\mathsf{Ps}(2) = \emptyset$, $\mathsf{Ps}(3) = \mathsf{Ps}(4) = \mathsf{Ps}(5) = \{N(y)\}$ (given by the inductive predicate formulas in sequents which are part of the inductive argument),
- R is given by the ancestry relationship between formulas in the proof rules:
  $\mathsf{R}_{0,1}(N(x),N(x),\rightsquigarrow)$, $\mathsf{R}_{1,3}(N(x),N(y),\searrow)$, $\mathsf{R}_{3,4}(N(y),N(y),\rightsquigarrow)$,
  $\mathsf{R}_{4,5}(N(y),N(y),\rightsquigarrow)$, $\mathsf{R}_{5,1}(N(y),N(x),\rightsquigarrow)$.

Note that a decrease ($\searrow$) only occurs at the application of a Split rule.

Example 2.12. The sloped graph corresponding to the cyclic preproof of Example 2.9 has the following components: vertices $\mathsf{V} = \{0, \ldots, 17\} \setminus \{7, 13, 17\}$ (all vertices except for buds); the edges are those of the preproof, with edges to the buds replaced by edges to the targets of their back-links, that is $\mathsf{E} = \{(0,1), (0,8), (1,2), (1,3), (3,4), (3,5), (5,6), (6,0), (8,9), (8,10), (10,11), (11,12), (12,0), (10,14), (14,15), (15,16), (16,0)\}$; Ps again gives all occurrences of formulas of the form $N(t)$ appearing in the sequent; and, again, R is given by the ancestry relationship between formulas in the proof rules, with decreases along applications of Split highlighted below:

$\mathsf{R}_{0,1}(N(x), N(x), \rightsquigarrow)$, $\mathsf{R}_{1,3}(N(x), N(x'), \leadsto)$, $\mathsf{R}_{3,5}(N(x'), N(\mathsf{s}(x'')), \rightsquigarrow)$, $\mathsf{R}_{3,5}(N(x'), N(x''), \leadsto)$,
$\mathsf{R}_{5,6}(N(\mathsf{s}(x'')), N(\mathsf{s}(x'')), \rightsquigarrow)$, $\mathsf{R}_{5,6}(N(x''), N(x''), \rightsquigarrow)$,
$\mathsf{R}_{6,0}(N(\mathsf{s}(x'')), N(x), \rightsquigarrow)$, $\mathsf{R}_{6,0}(N(x''), N(y), \rightsquigarrow)$, $\mathsf{R}_{8,9}(N(x), N(x), \rightsquigarrow)$,
$\mathsf{R}_{8,10}(N(x), N(x), \rightsquigarrow)$, $\mathsf{R}_{8,10}(N(y'), N(\mathsf{s}(y'')), \rightsquigarrow)$, $\mathsf{R}_{8,10}(N(y'), N(y''), \leadsto)$,
$\mathsf{R}_{10,11}(N(\mathsf{s}(y'')), N(\mathsf{s}(y'')), \rightsquigarrow)$, $\mathsf{R}_{10,11}(N(y''), N(y''), \rightsquigarrow)$,
$\mathsf{R}_{11,12}(N(\mathsf{s}(y'')), N(\mathsf{s}(y'')), \rightsquigarrow)$, $\mathsf{R}_{11,12}(N(y''), N(y''), \rightsquigarrow)$,
$\mathsf{R}_{12,0}(N(\mathsf{s}(y'')), N(x), \rightsquigarrow)$, $\mathsf{R}_{12,0}(N(y''), N(y), \rightsquigarrow)$,
$\mathsf{R}_{10,14}(N(x), N(x'), \leadsto)$, $\mathsf{R}_{10,14}(N(\mathsf{s}(y'')), N(\mathsf{s}(y'')), \rightsquigarrow)$, $\mathsf{R}_{10,14}(N(y''), N(y''), \rightsquigarrow)$,
$\mathsf{R}_{14,15}(N(x'), N(x'), \rightsquigarrow)$, $\mathsf{R}_{14,15}(N(\mathsf{s}(y'')), N(\mathsf{s}(y'')), \rightsquigarrow)$, $\mathsf{R}_{14,15}(N(y''), N(y''), \rightsquigarrow)$,
$\mathsf{R}_{15,16}(N(x'), N(x'), \rightsquigarrow)$, $\mathsf{R}_{15,16}(N(y''), N(y''), \rightsquigarrow)$, $\mathsf{R}_{16,0}(N(x'), N(x), \rightsquigarrow)$, $\mathsf{R}_{16,0}(N(y''), N(y), \rightsquigarrow)$.

Note that, in the Split rule, only the active formulas introduce decreases, e.g., we have $\mathsf{R}_{8,10}(N(y'), N(y''), \leadsto)$, but $\mathsf{R}_{8,10}(N(x), N(x), \rightsquigarrow)$ and $\mathsf{R}_{8,10}(N(y'), N(\mathsf{s}(y'')), \rightsquigarrow)$. Indeed, at vertex 8, the rule $\textsc{Split}_N$ was applied (backwards) to the sequent $\Gamma, N(y') \vdash H(x, s(y'))$ where $\Gamma = N(x), N(y')$, yielding $\Gamma[\mathsf{s}(y'')/y'], N(y'') \vdash H(x, \mathsf{s}(y'))[\mathsf{s}(y'')/y']$ for the right-hand premise, i.e., $N(x), N(\mathsf{s}(y'')), N(y'') \vdash H(x, \mathsf{s}(\mathsf{s}(y'')))$. (Since sequents are comprised of *sets*, here $\Gamma, N(y')$ is equal to Γ.)

Our abstraction via sloped graphs was partly inspired by the automata construction for deciding Infinite Descent in [Brotherston 2006], and is a further generalisation of the abstract account of cyclic proof systems given in [Brotherston 2006; Brotherston et al. 2012] (e.g. the $(\mathsf{Pos}, \mathsf{V})$ component of SG corresponds to the notion of "preproof graph" [Brotherston 2006, Def. 5.1.3]). Recently, Afshari and Wehr [2022] have also proposed an abstract, category-theoretic description of cyclic proofs.

For a sloped graph, we use standard graph-theoretic concepts such as finite and infinite paths, $\pi = (v_i)_{i<n}$ and $(v_i)_{i\in\mathbb{N}}$, respectively; we call the latter *ipaths*. We call a finite path $\pi = (v_i)_{i<n}$ *non-trivial* when $n > 1$. We write $\mathsf{Path}_{v,v'}$ for the set of paths between v and v', and $\mathsf{IPath}(\mathsf{SG})$ for the set of infinite paths in SG. If $\pi \in \mathsf{Path}_{v,v'}$ and $\pi' \in \mathsf{Path}_{v',v''}$, we write $\pi \cdot \pi'$ for the composition of the two.

Definition 2.13. A *trace for an ipath* $(v_i)_{i\in\mathbb{N}}$ in SG is an infinite sequence $(p_i)_{i\in\mathbb{N}}$ of positions such that the following hold for all $i \in \mathbb{N}$: (1) $p_i \in \mathsf{Ps}(v_i)$ and (2) $\exists s \in \mathsf{S}.\, \mathsf{R}_{v_i,v_{i+1}}(p_i, p_{i+1}, s)$. We say the trace is *decreasing* if for all $i \in \mathbb{N}$ there exists $j \geq i$ such that $\mathsf{R}_{v_j,v_{j+1}}(p_j, p_{j+1}, \leadsto)$. An ipath $(v_i)_{i\in\mathbb{N}}$ is said to be *descending* if it has a tail (i.e., $(v_i)_{i\geq i_0}$, for some $i_0 \in \mathbb{N}$) with a decreasing trace.

Definition 2.14. A sloped graph SG is said to satisfy *Infinite Descent* if all its ipaths are descending.

In summary: Infinite Descent says that, every ipath in the graph has a decreasing trace starting from some i_0, i.e., a choice of positions along it such that the slope between any consecutive positions is *always maintained or decreased* and is *always eventually decreased*.

So in a sloped graph the positions are the items that can be traced by Infinite Descent, which in our FOLID-based examples are the inductive predicates occurring in any antecedent of a sequent in the proof. The vertex width of a sloped graph (the maximum number of positions per node) in some sense captures the "local size" of the Infinite Descent Problem—a central parameter in our complexity analysis. For the sloped graphs in Examples 2.11 and 2.12, the vertex width is 1 and 3 respectively, the latter because, e.g., node 10 has positions 3 positions: $N(x)$, $N(\mathsf{s}(y''))$ and $N(y'')$.

To obtain the Size-Change Principle [Lee et al. 2001] instead of Infinite Descent, we can instantiate our abstract sloped graph setting by taking the vertices to be function calls, the positions to be function parameters and the position changes to be known orderings between these parameters. In this instantiation, their *size-change graphs* between two function names are sloped relations between their parameters (together with their domains and codomains), and their fixed set of size-change graphs associated to a subject program forms a sloped graph; their *multi-paths* correspond to our paths and ipaths (but having the sloped relations attached); their *threads* in a multipath correspond to (finite or infinite) prefixes of our traces for an ipath. (A more direct representation of their setting would use sloped *multi*-graphs, where the edges are function calls.)

For a sloped graph it is often possible to systematically reduce the number of vertices whilst preserving its 'topology', and thus its Infinite Descent Property. That is, the reduced sloped graph satisfies Infinite Descent iff the original one does. The Cyclist theorem prover [Brotherston et al. 2012] does this as part of its pipeline; Afshari and Wehr [2022] describe and prove correct an abstract reduction procedure. Such reductions proceed by: (i) the restriction to strongly connected components (SCCs) only; (ii) collapsing paths into single edges, where this does not lead to identifying distinct cycles; and (iii) the composition of the sloped relations along such collapsed paths.

Example 2.15. A minimal reduced version $(\mathsf{V}', \mathsf{E}', \mathsf{Ps}', \mathsf{R}')$ of the sloped graph $(\mathsf{V}, \mathsf{E}, \mathsf{Ps}, \mathsf{R})$ from Example 2.11 has the following components: $\mathsf{V}' = \{1\}$, $\mathsf{E}' = \{(1,1)\}$, $\mathsf{Ps}'(1) = \{N(x)\}$, and R' is given by $\mathsf{R}_{1,1}(N(x), N(x), \searrow)$. Note that the above was obtained from Example 2.11's graph as follows: (i) the vertices 0 and 2 have been removed since they are outside the graph's only SCC, (ii) the cycle $1 \cdot 3 \cdot 4 \cdot 5 \cdot 1$ was collapsed into the self-edge $(1,1)$, and (iii) the relation $\mathsf{R}'_{1,1}$ was defined as the composition of the sloped relations from Example 2.11's collapsed cycle, $\mathsf{R}_{1,3} \circ \mathsf{R}_{3,4} \circ \mathsf{R}_{4,5} \circ \mathsf{R}_{5,1}$.

Example 2.16. We define $(\mathsf{V}', \mathsf{E}', \mathsf{Ps}', \mathsf{R}')$, a minimal reduced version of the sloped graph $(\mathsf{V}, \mathsf{E}, \mathsf{Ps}, \mathsf{R})$ from Example 2.12, by: $\mathsf{V}' = \{0, 11, 14\}$; $\mathsf{E}' = \{(0,0), (0,11), (11,0), (0,14), (14,0)\}$; $\mathsf{Ps}'(0) = \{N(x), N(y)\}$, $\mathsf{Ps}'(11) = \{N(\mathsf{s}(y'')), N(y'')\}$, $\mathsf{Ps}'(14) = \{N(x'), N(\mathsf{s}(y'')), N(y'')\}$; and R' given by:
$\mathsf{R}'_{0,0}(N(x), N(x), \searrow)$, $\mathsf{R}'_{0,0}(N(x), N(y), \searrow)$,
$\mathsf{R}'_{0,11}(N(y), N(\mathsf{s}(y'')), \searrow)$, $\mathsf{R}'_{0,11}(N(y), N(y''), \searrow)$, $\mathsf{R}'_{11,0}(N(\mathsf{s}(y'')), N(x), \rightsquigarrow)$, $\mathsf{R}'_{11,0}(N(y''), N(y), \rightsquigarrow)$,
$\mathsf{R}'_{0,14}(N(x), N(x'), \searrow)$, $\mathsf{R}'_{0,14}(N(y), N(\mathsf{s}(y'')), \searrow)$, $\mathsf{R}'_{0,14}(N(y), N(y''), \searrow)$,
$\mathsf{R}'_{14,0}(N(x'), N(x), \rightsquigarrow)$, $\mathsf{R}'_{14,0}(N(y''), N(y), \rightsquigarrow)$.
The above was obtained from Example 2.12's graph as follows:

(i) the nodes 2, 4, 9 have been removed since they are outside the graph's only SCC,
(ii) the cycle $0 \cdot 1 \cdot 3 \cdot 5 \cdot 6 \cdot 0$ was collapsed into the self-edge $(0,0)$, the paths $0 \cdot 8 \cdot 10 \cdot 11$ and $11 \cdot 12 \cdot 0$ were collapsed into the edges $(0,11)$ and $(11,0)$ respectively, and the paths $0 \cdot 8 \cdot 10 \cdot 14$ and $14 \cdot 15 \cdot 16 \cdot 0$ were collapsed into the edges $(0,14)$ and $(14,0)$ respectively;
(iii) the sloped relations are defined as compositions of those along the paths of the original graph, e.g., $\mathsf{R}_{0,0}$ is $\mathsf{R}_{0,1} \circ \mathsf{R}_{1,3} \circ \mathsf{R}_{3,5} \circ \mathsf{R}_{5,6} \circ \mathsf{R}_{6,0}$, and $\mathsf{R}_{0,11}$ is $\mathsf{R}_{0,8} \circ \mathsf{R}_{8,10} \circ \mathsf{R}_{10,11}$.

There is no unique way to produce a minimal reduced version. Here, we prioritized the full collapsing of the left-most simple cycle to produce the self-edge $(0,0)$, which meant that the collapse of the other two simple cycles each had to be mediated by at least one other vertex. We could have gone other ways, e.g., fully collapsing the middle or right-most simple cycles. These cycles must remain distinct after the collapse, because we need to preserve distinct sloped relations along them.

3 OVERVIEW OF THE COMPLEXITY LANDSCAPE

In the remainder of the paper, we explore existing and novel criteria for checking Infinite Descent for sloped graphs. Namely, we study the following criteria, where the first two are automaton-based and the last two are relation-based.

Criterion	Time Complexity Upper Bound
Vertex-language Automaton	$O(n^5 \cdot w^2 \cdot 2^{2nw\log(2nw)})$
Slope-language Automaton	$O(n^2 \cdot w \cdot \mathsf{Min}(n^4, 3^{2w^2}) \cdot 2^{2w\log(2w)})$
Transitive Looping (SCT)	$O(n{\cdot}(w^4{\cdot}3^{3w^2} + n^2{\cdot}w^4{\cdot}3^{2w^2}))$
Order-reduced Transitive Looping	$O(n^3 \cdot w^4 \cdot 3^{2w^2})$

Fig. 3. Time complexity bounds for the various criteria

Vertex-language Automaton criterion: the most common criterion for Infinite Descent, based on automata recognising vertices of a sloped graph (see Section 4.1).

Slope-language Automaton criterion: a novel automaton-based criterion that shifts from an alphabet of vertices to one of sloped relations (see Section 4.2).

Transitive Looping criterion: essentially, the size-change termination (SCT) criterion given in [Lee et al. 2001] (see Section 5.1).

Order-reduced Transitive Looping criterion: a novel relation-based criterion for Infinite Descent, inspired by and improving on the Transitive Looping criterion (see Section 5.2).

Sections 4 and 5 of this paper provide a comprehensive examination of the aforementioned criteria, offering a detailed parameterised analysis of their worst-case time complexity in terms of the number of vertices n in the sloped graph and the vertex width w (see Definition 2.10).

Our complexity results are summarized in Fig. 3. The Vertex-language Automaton exhibits exponential complexity with respect to both n and w, whereas the Slope-language Automaton is exponential only in the latter and polynomial in the former. The relation-based approaches demonstrate a clear distinction between the two parameters, showcasing polynomial complexity in the number of vertices and exponential complexity in the vertex width. Our novel order-reduced criterion improves upon the complexity bounds of existing relation-based criteria, reducing the dependence on the vertex width by an exponential factor. Notably, though, there is no definitive winner between the Order-reduced Transitive Looping criterion and the automaton-based criteria, in terms of time complexity bounds. The superiority of a criterion depends on the relationship between the two parameters. When w is comparable to n, the exponential component of the Vertex-language Automaton criterion is similar to that of Order-reduced Transitive Looping, but the latter has a slightly better polynomial profile. If w dominates n then, complexity-wise, the automata-based criteria prevail (with the Slope-language Automaton criterion having slightly better complexity). When n is exponentially larger than w, the Order-reduced Transitive Looping criterion has by far the best theoretical complexity. Our experimental evaluation of these methods presented in Section 6 bear out the observation that the various criteria behave quite diferently depending on the context, and whether or not the input graphs satisfy Infinite Descent. Altogether this shows that these different trade-offs between the methods create an intriguing complexity landscape.

4 AUTOMATON-BASED CRITERIA

Perhaps the most well-known characterization of the Infinite Descent property is in terms of inclusion between ω-regular languages, i.e., those accepted by Büchi automata. This originated with Lee et al. [2001] in the context of the Size-Change Principle for program termination, and was later adapted to the logical context for Infinite Descent [Brotherston 2006; Simpson 2017].

We describe the method using our framework's terminology, augmenting with (time) complexity bounds, and we propose an alternative automaton-theoretic method. We adopt standard definitions

for Büchi automata $\mathcal{B} = (\Sigma, Q, q_0, \Delta, F)$, consisting of an alphabet Σ, a set of states Q with initial state $q_0 \in Q$ and set $F \subseteq Q$ of accepting states, and a transition relation $\Delta \subseteq Q \times \Sigma \times Q$. A *word* over the alphabet Σ is an infinite sequence of letters $w = (a_i)_{i \in \mathbb{N}}$. A *run* for $w = (a_i)_{i \in \mathbb{N}}$ is an infinite sequence of states $(s_i)_{i \in \mathbb{N}}$ such that $s_0 = q_0$ and $(s_i, a_i, s_{i+1}) \in \Delta$ for all $i \in \mathbb{N}$. A run $(s_i)_{i \in \mathbb{N}}$ for w is called *accepting* if it visits accepting states infinitely often, that is, the set $\{i \in \mathbb{N} \mid s_i \in F\}$ is infinite. A word w is said to be *accepted* by $\mathcal{B}$ if it has an accepting run. The *language of* $\mathcal{B}$, denoted by $\mathsf{Lang}(\mathcal{B})$, is the set of words accepted by $\mathcal{B}$. See [Khoussainov and Nerode 2001] for a comprehensive treatment. The standard procedure for deciding the emptiness of $\mathsf{Lang}(\mathcal{B})$ operates by checking if there exists a strongly connected component in $\mathcal{B}$'s underlying directed graph that is accessible from the initial state and contains a final state; the worst-case time complexity for this procedure equals that of determining the strongly connected components, which is the size of this underlying graph [Tarjan 1972], approximated from above by $|Q| + |\Delta|$.

4.1 Vertex-language Automaton Criterion

The construction of Lee et al. [2001] uses Büchi automata recognising words over the alphabet $\Sigma = \mathsf{V}$ of vertices in the (proof) graph. We follow Brotherston's [2006] account. Given a sloped graph $\mathsf{SG} = (\mathsf{V}, \mathsf{E}, \mathsf{Ps}, \mathsf{R})$, we construct two Büchi automata, both over the alphabet $\Sigma = \mathsf{V}$:

- a *path automaton*, $\mathsf{PAut}_\mathsf{V}(\mathsf{SG})$, accepting all ipaths in SG, *i.e.* $\mathsf{Lang}(\mathsf{PAut}_\mathsf{V}(\mathsf{SG})) = \mathsf{IPath}(\mathsf{SG})$;
- a *trace automaton*, $\mathsf{TAut}_\mathsf{V}(\mathsf{SG})$, accepting all infinite sequences for which some suffix has a decreasing trace: $\mathsf{Lang}(\mathsf{TAut}_\mathsf{V}(\mathsf{SG})) = \{(v_i)_{i \in \mathbb{N}} \mid \exists i_0 \in \mathbb{N}.\ (p_i)_{i \geq i_0} \textit{ decreasing trace for } (v_i)_{i \geq i_0}\}$.

The path automaton $\mathsf{PAut}_\mathsf{V}(\mathsf{SG}) = (\Sigma, Q, q_0, \Delta, F)$ is immediate: $Q = \mathsf{V}$, q_0 is the root vertex of the proof, $\Delta = \{(v, v', v') \mid (v, v') \in \mathsf{E}\}$, and $F = \mathsf{V}$. The transitions from each vertex v allow to choose a vertex v' to which v is connected by an edge in SG, and the target vertex is recognised at each such transition.

The trace automaton $\mathsf{TAut}_\mathsf{V}(\mathsf{SG}) = (\Sigma, Q', q_0', \Delta', F')$ is defined as follows:

- $Q' = \{q_0'\} \cup \{(v, p, s) \mid v \in \mathsf{V} \wedge p \in \mathsf{Ps}(v) \wedge s \in \mathsf{S}\}$, with q_0' some fresh initial state.
- $\Delta' = \Delta_1 \cup \Delta_2 \cup \Delta_3$, where $\Delta_1 = \{(q_0', v, q_0') \mid v \in \mathsf{V}\}$, $\Delta_2 = \{(q_0', v, (v, p, s)) \mid (v, p, s) \in Q'\}$, and $\Delta_3 = \{((v, p, s), v', (v', p', s')) \mid s \in \mathsf{S} \wedge (v, v') \in \mathsf{E} \wedge \mathsf{R}_{v,v'}(p, p', s')\}$
- $F = \{(v, p, \lightning) \mid (v, p, \lightning) \in Q'\}$

Remaining within the initial state, q_0', any number of arbitrary transitions are allowed (the Δ_1 part). At some nondeterministic point on the ipath, tracking the descent begins by moving into a state that stores not only the vertex, but also a position of that vertex and the last occurring slope relationship between positions (the Δ_2 part). Thereafter, a "maintain" or "decrease" slope relationship has to be continuously guaranteed between the positions (the Δ_3 part). Note that in this automaton any sequence of nodes $(v_i)_{i \in \mathbb{N}}$ has a run using only transitions in Δ_1. However, the acceptance condition F forces an (accepting) run to (1) reach transitions in Δ_3 (via Δ_2), where it then remains, and for (2) infinitely often these Δ_3 transitions to represent a "decrease". Thus, a word $(v_i)_{i \in \mathbb{N}}$ is accepted by $\mathsf{TAut}_\mathsf{V}(\mathsf{SG})$ iff some suffix of it $(v_i)_{i \geq i_0}$ has a decreasing trace $(p_i)_{i \geq i_0}$.

Since Infinite Descent means that all ipaths are descending, it is reducible to the inclusion between the languages of the above two automata:[1]

Theorem 4.1 (Vertex-language Automaton criterion, [Brotherston 2006; Lee et al. 2001; Simpson 2017]). *A sloped graph* SG *satisfies the Infinite Descent property if and only if* $\mathsf{Lang}(\mathsf{PAut}_\mathsf{V}(\mathsf{SG})) \subseteq \mathsf{Lang}(\mathsf{TAut}_\mathsf{V}(\mathsf{SG}))$.

Inclusion between two Büchi automata $\mathcal{B}$ and $\mathcal{B}'$, can be decided in time exponential in the combined sizes of $\mathcal{B}$ and $\mathcal{B}'$. The standard algorithm checks emptiness of the language of $\mathcal{B} \times \overline{\mathcal{B}'}$,

[1] Full details on this construction can be found in [Brotherston 2006, App. A].

the product between $\mathcal{B}$ and the complement of $\mathcal{B}'$ (but there are also more efficient ones [Abdulla et al. 2010]). The best known bound for complementing a Büchi automata $\mathcal{B}$ with k states is given by Schewe's [2009] ranked-based approach, yielding an automaton $\overline{\mathcal{B}}$ with $O(2^{k \cdot \log k})$ states. Standard complexity analyses posit that this dominates the time of both constructing the complement automaton, and the overall inclusion check. However, this assumes a fixed alphabet. By contrast, here we wish to distinguish different parameters (affecting, in different ways, the sizes of the automaton components), and in our case, the alphabet is not fixed but proportional to the input. So we take as a lower bound on the time complexity the size not only of the set of states, but also that of the transition relation of Schewe's complement automaton—this is in $O(2^{|Q| \cdot \log |Q|} \cdot |\Sigma| \cdot |\Delta|)$. We also then consider the states and transition relation of the product automaton.

Let $n = |\mathsf{V}|$ and w be the vertex width of SG. From the equivalence in Theorem 4.1 we can derive:

Theorem 4.2. *The Vertex-language Automaton criterion is decidable in time* $O(n^5 \cdot w^2 \cdot 2^{2nw \log(2nw)})$.

Proof. Let us recall the sizes of the components of the automata. We have that $|\Sigma| = n$. Let $m = |\mathsf{E}|$. For $\mathsf{PAut}_\mathsf{V}(\mathrm{SG})$ we have $|Q| = n$ and $|\Delta| = m$. For $\mathsf{TAut}_\mathsf{V}(\mathrm{SG})$ we have: $|Q'| = 2nw + 1$, $|\Delta_1| = n$, $|\Delta_2| = 2nw + 1$, $|\Delta_3| \leq 2mw^2 \leq 2n^2w^2$, and $|\Delta'| = |\Delta_1| + |\Delta_2| + |\Delta_3| = O(n^2w^2)$, where we have approximated m as $O(n^2)$. As mentioned, the sizes of the components of the complement automaton for $\mathsf{TAut}_\mathsf{V}(\mathrm{SG})$ are $|\overline{Q'}| = O(2^{|Q'| \log |Q'|}) = O(2^{2nw \log(2nw)})$ for the states, and $|\overline{\Delta'}| = O(2^{|Q'| \log |Q'|} \cdot |\Sigma| \cdot |\Delta'|) = O(n^3 \cdot w^2 \cdot 2^{2nw \log(2nw)})$ for the transition relation. Finally, we compute the product automaton between the path automaton and the complement of the trace automaton using the standard construction for Büchi automata. This yields a set of states whose size is (up to a constant factor) the product of the sizes of the sets of states of the input automata: $|Q| \cdot |\overline{Q'}| = O(n \cdot 2^{2nw \log(2nw)})$. Also, in the worst case, the size of the transition relation is (up to a constant factor) the product of the sizes of the transition relations of the input automata: $|\Delta| \cdot |\overline{\Delta'}| = O(n^5 \cdot w^2 \cdot 2^{2nw \log(2nw)})$. The overall complexity is obtained by taking the last number above, which subsumes all others. □

4.2 Slope-language Automaton Criterion

The complexity of the Vertex-language Automaton criterion is exponential in both the parameters n and w. Due to the PSPACE-completeness of the problem, we cannot expect to eliminate exponential behaviour altogether. However, an interesting question is whether we can remove the exponentiality in one of the parameters, perhaps with a trade-off of further complexity in the other. Next, we propose a novel automaton-theoretic solution that achieves this. (In Section 5 we explore approaches based on relational abstractions that also achieve this.)

For the Vertex-language criterion, the exponentiality in n comes from the complementation of the trace automaton. Next, we will focus on removing the dependency of the trace automaton on n—by noting that, in the incremental construction of a (decreasing) trace for an ipath, what matters is not the particular current vertex or its out-edges but, more abstractly, the sloped relations on the edges.

This suggests shifting from an alphabet of vertices to one of sloped relations. We will only need those that appear in SG, so we take $\Sigma = \{\mathsf{R}_{v,v'} \mid (v, v') \in \mathsf{E}\}$. We will construct the path and trace automata such that only the path automaton "knows" about vertices/edges. Its construction is very similar to vertex-language path automaton, except that it recognises sloped relations along edges instead of the target vertices. Namely, we define the path automaton $\mathsf{PAut}_\mathsf{R}(\mathrm{SG}) = (\Sigma, Q, q_0, \Delta, F)$ taking: $Q = \{q_0\} \cup \mathsf{V}$ where q_0 is some fresh initial state, $\Delta = \{(q_0, \mathsf{R}_{v,v'}, v') \mid (v, v') \in \mathsf{E}\} \cup \{(v, \mathsf{R}_{v,v'}, v') \mid (v, v') \in \mathsf{E}\}$, and $F = \mathsf{V}$. Therefore, $\mathsf{PAut}_\mathsf{R}(\mathrm{SG})$ accepts all infinite sequences $(R_i)_{i \in \mathbb{N}}$ of sloped relations such that there is an ipath $(v_i)_{i \in \mathbb{N}}$ in SG satisfying $R_j = \mathsf{R}_{v_j, v_{j+1}}$ for all $j \in \mathbb{N}$.

Conversely, the trace automaton $\mathsf{TAut}_\mathsf{R}(\mathrm{SG}) = (\Sigma, Q', q_0', \Delta', F')$ only "remembers" positions and slopes, transitioning from one position to another via sloped relations that permit this:

- $Q' = \{q'_0\} \cup \{(p, s) \mid p \in (\bigcup_{v \in V} \mathsf{Ps}(v)) \wedge s \in S\}$ with q'_0 some fresh initial state
- $\Delta' = \Delta_1 \cup \Delta_2 \cup \Delta_3$, where $\Delta_1 = \{(q'_0, R, q'_0) \mid R \in \Sigma\}$,
  $\Delta_2 = \{(q'_0, R, (p', s')) \mid R \in \Sigma \wedge (p', s') \in Q'\}$, and
  $\Delta_3 = \{((p, s), R, (p', s')) \mid (p, s) \in Q' \wedge R \in \Sigma \wedge R(p, p', s')\}$
- $F = \{(p, \rightsquigarrow) \mid (p, \rightsquigarrow) \in Q'\}$

The Δ_1 and Δ_2 transitions play a similar role as in the vertex-language trace automaton: allowing to delay the start of tracing the descent. Note that the runs of the automaton need correspond to any actual path in SG. Formally, a word $(R_i)_{i \in \mathbb{N}}$ is accepted by $\mathsf{TAut}_R(SG)$ when there exists some suffix $(R_i)_{i \geq i_0}$ and a sequence $(p_i)_{i \geq i_0}$ of positions such that, for all $i \geq i_0$: $\exists s \in S.\, R_i(p_i, p_{i+1}, s)$ and $\exists j \geq i.\, R_j(p_j, p_{j+1}, \rightsquigarrow)$. Thus, its accepting runs correspond to a form of path-less decreasing trace.

While recognising sloped relations (not vertices), these automata also encode Infinite Descent:

Theorem 4.3 (Slope-language Automaton criterion). *A sloped graph* SG *satisfies the Infinite Descent property iff* $\mathsf{Lang}(\mathsf{PAut}_R(SG)) \subseteq \mathsf{Lang}(\mathsf{TAut}_R(SG))$.

Proof Outline. The result derives from the following two facts, straightforwardly verified:

1. For any sequence $(R_i)_{i \in \mathbb{N}}$, we have that $(R_i)_{i \in \mathbb{N}} \in \mathsf{Lang}(\mathsf{PAut}_R(SG))$ iff there exists an ipath $(v_i)_{i \in \mathbb{N}}$ such that $\forall i \in \mathbb{N}.\, R_i = \mathsf{R}_{v_i, v_{i+1}}$;
2. For any ipath $(v_i)_{i \in \mathbb{N}}$, we have that $(\mathsf{R}_{v_i, v_{i+1}})_{i \in \mathbb{N}} \in \mathsf{Lang}(\mathsf{TAut}_R(SG))$ iff $(v_i)_{i \in \mathbb{N}}$ is descending.

Now the proof of the desired fact goes as follows:

(if): First assume $\mathsf{Lang}(\mathsf{PAut}_R(SG)) \subseteq \mathsf{Lang}(\mathsf{TAut}_R(SG))$. To show that Infinite Descent holds, let $(v_i)_{i \in \mathbb{N}}$ be an ipath. From point 1, we have that $(\mathsf{R}_{v_i, v_{i+1}})_{i \in \mathbb{N}} \in \mathsf{Lang}(\mathsf{PAut}_R(SG))$, and therefore $(\mathsf{R}_{v_i, v_{i+1}})_{i \in \mathbb{N}} \in \mathsf{Lang}(\mathsf{TAut}_R(SG))$. Hence, from point 2 we have that $(v_i)_{i \in \mathbb{N}}$ is descending.

(only if): Now, assume that Infinite Descent holds, and let $(R_i)_{i \in \mathbb{N}} \in \mathsf{Lang}(\mathsf{PAut}_R(SG))$. By point 1 above, we obtain an ipath $(v_i)_{i \in \mathbb{N}}$ such that $\forall i \in \mathbb{N}.\, R_i = \mathsf{R}_{v_i, v_{i+1}}$. From Infinite Descent, we have that $(v_i)_{i \in \mathbb{N}}$ is descending. Hence, from point 2 we have $(\mathsf{R}_{v_i, v_{i+1}})_{i \in \mathbb{N}} \in \mathsf{Lang}(\mathsf{TAut}_R(SG))$, i.e., $(R_i)_{i \in \mathbb{N}} \in \mathsf{Lang}(\mathsf{TAut}_R(SG))$. □

The slope-language constructions achieve the goal of reducing the exponentiality in the number of vertices, and indeed this happens at the expense of added complexity in the vertex width (unless n is exponentially larger). From the equivalence in Theorem 4.3 we can derive:

Theorem 4.4. *The Slope-language Automaton criterion is decidable in time* $O(n^2 \cdot w \cdot \mathsf{Min}(n^4, 3^{2w^2}) \cdot 2^{2w \log(2w)})$.

Proof. The total number of sloped relations between sets not exceeding w elements is at most 3^{w^2}. However, note that SG has at most $m = |E|$ distinct sloped relations (one for each edge). Thus we have $|\Sigma| = O(\mathsf{Min}(n^2, 3^{w^2}))$, since $m = O(n^2)$. Then, for $\mathsf{PAut}_R(SG)$ we have $|Q| = n + 1$ and $|\Delta| = m + m = O(n^2)$. Moreover, note that, given a particular sloped graph, we can assume without loss of generality that $|\mathsf{Pos}| = w$ (by renaming the positions if necessary). Thus, we only need at most w positions. So, for $\mathsf{TAut}_R(SG)$ we have $|Q'| = 2w + 1$, $|\Delta_1| = \mathsf{Min}(m, 3^{w^2})$, $|\Delta_2| = |\Delta_3| = (|Q'| - 1) \cdot \mathsf{Min}(m, 3^{w^2}) = 2w \cdot \mathsf{Min}(m, 3^{w^2})$, and thus $|\Delta'| = |\Delta_1| + |\Delta_2| + |\Delta_3| = O(w \cdot \mathsf{Min}(n^2, 3^{w^2}))$. As before, the sizes of the components of the complement automaton for $\mathsf{TAut}_R(SG)$ are $|\overline{Q'}| = O(2^{|Q'| \cdot \log |Q'|}) = O(2^{2w \log(2w)})$ for the states, and for the transition relation we have $|\overline{\Delta'}| = O(2^{|Q'| \cdot \log |Q'|} \cdot |\Sigma| \cdot |\Delta'|) = O(w \cdot \mathsf{Min}(n^4, 3^{2w^2}) \cdot 2^{2w \log(2w)})$. Finally, the product of the path automaton and the complement of the trace automaton needs a set of states whose size is proportional to $|Q| \cdot |\overline{Q'}| = O(n \cdot 2^{2w \log(2w)})$, and a transition relation whose size is proportional to $|\Delta| \cdot |\overline{\Delta'}| = O(n^2 \cdot w \cdot \mathsf{Min}(n^4, 3^{2w^2}) \cdot 2^{2w \log(2w)})$. The overall complexity is obtained by taking the last number above, which subsumes all others. □

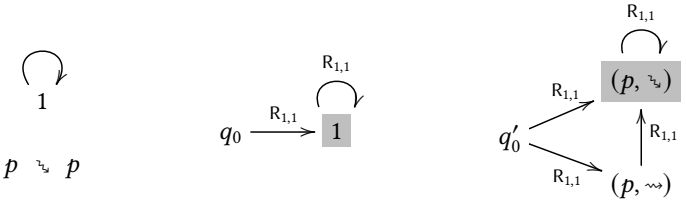


Fig. 4. The sloped graph (left), path automaton (middle) and trace automaton (right) for Example 4.5, with accepting states shaded in grey.

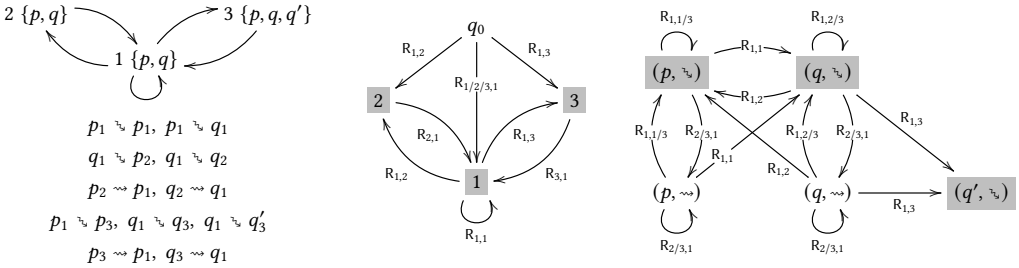


Fig. 5. The sloped graph (left), path automaton (middle) and trace automaton (right) for Example 4.6, with accepting states shaded in grey (the trace automaton's initial state, q'_0, and associated transitions, are elided).

Example 4.5. Fig. 4 shows Example 2.15's sloped graph SG (writing p for $N(x)$ and $p \searrow p$ for $\mathsf{R}_{1,1}(p, p, \searrow)$), and its slope-language path and trace automata $\mathsf{PAut}_{\mathsf{R}}(\mathrm{SG})$ and $\mathsf{TAut}_{\mathsf{R}}(\mathrm{SG})$. Since $\mathsf{PAut}_{\mathsf{R}}(\mathrm{SG})$ accepts a single word, $(\mathsf{R}_{1,1})^\omega$, which is accepted by $\mathsf{TAut}_{\mathsf{R}}(\mathrm{SG})$ via the run $q'_0(p, \searrow)^\omega$, we have $\mathsf{Lang}(\mathsf{TAut}_{\mathsf{R}}(\mathrm{SG})) \subseteq \mathsf{Lang}(\mathsf{PAut}_{\mathsf{R}}(\mathrm{SG}))$, reflecting that Infinite Descent holds for SG.

Example 4.6. A sloped graph SG equivalent (via some renaming) with the reduced 2-hydra one from Example 2.16 is shown in Fig. 5, where we show the sets of positions next to each node (e.g., $\mathrm{Ps}(1) = \{p, q\}$) and use suggestive notations for the sloped relations (e.g., $p_1 \searrow q_2$ means $\mathsf{R}_{1,2}(p, q, \searrow)$). The figure also shows the slope-language path and trace automata, $\mathsf{PAut}_{\mathsf{R}}(\mathrm{SG})$ and $\mathsf{TAut}_{\mathsf{R}}(\mathrm{SG})$, respectively, where we use '/' to indicate multiple labels, e.g., $\mathsf{R}_{2/3,1}$ denotes two labels: $\mathsf{R}_{2,1}$ and $\mathsf{R}_{3,1}$. Note that $\mathsf{PAut}_{\mathsf{R}}(\mathrm{SG})$ accepts the language $((\mathsf{R}_{1,1})^*\ (\mathsf{R}_{1,2}\,\mathsf{R}_{2,1})^*\ (\mathsf{R}_{1,3}\,\mathsf{R}_{3,1})^*)^\omega$. Each word in this language has one of the following two forms, depending on whether it ends in a $(\mathsf{R}_{1,3}\,\mathsf{R}_{3,1})^\omega$ loop: either (i) $(\prod_{i<h} w_i)\,(\mathsf{R}_{1,3}\,\mathsf{R}_{3,1})^\omega$ with $h \in \mathbb{N}$ or (ii) $\prod_{i\in\mathbb{N}} w_i$, where each w_i is either (a) $(\mathsf{R}_{1,1})^{k_i}\,(\mathsf{R}_{1,3}\,\mathsf{R}_{3,1})^{l_i}$, or (b) $(\mathsf{R}_{1,2}\,\mathsf{R}_{2,1})^{k_i}\,(\mathsf{R}_{1,3}\,\mathsf{R}_{3,1})^{l_i}$, for $k_i \in \mathbb{N}_+$ and $l_i \in \mathbb{N}$. (It is not required that the forms (a) and (b) actually alternate in the sequence of w_i's.)

If the word has the form (i), then it is accepted by $\mathsf{TAut}_{\mathsf{R}}(\mathrm{SG})$ via the run $(q'_0)^t((p, \searrow)\,(p, \rightsquigarrow))^\omega$, where $t = \sum_{i<h} \mathsf{length}(w_i)$. If the word has the form (ii), then it is accepted by $\mathsf{TAut}_{\mathsf{R}}(\mathrm{SG})$ via the run $q'_0\,(\prod_{i\in\mathbb{N}} u_i)$, where each u_i is the following sequence of $\mathsf{TAut}_{\mathsf{R}}(\mathrm{SG})$ states:

- $(p, \searrow)^{k_i}\,((p, \searrow)\,(p, \rightsquigarrow))^{l_i}$, if w_i has the form (a) and w_{i+1} also has the form (a);
- $(p, \searrow)^{k_i-1}\,(q, \searrow)\,((q, \searrow)\,(q, \rightsquigarrow))^{l_i}$, if w_i has the form (a) and w_{i+1} has the form (b);
- $((q, \searrow)\,(q, \rightsquigarrow))^{k_i}\,((q, \searrow)\,(q, \rightsquigarrow))^{l_i}$, if w_i has the form (b) and w_{i+1} also has the form (b);
- $((q, \searrow)\,(q, \rightsquigarrow))^{k_i-1}\,(p, \searrow)\,((p, \searrow)\,(p, \rightsquigarrow))^{l_i}$, if w_i has the form (b) and w_{i+1} has the form (a).

So, the states that make up the run shift between $(p, \searrow)$ and $(q, \searrow)$, depending on the next cycle in which the path-following word engages. Thus $\mathsf{Lang}(\mathsf{TAut}_{\mathsf{R}}(\mathrm{SG})) \subseteq \mathsf{Lang}(\mathsf{PAut}_{\mathsf{R}}(\mathrm{SG}))$, reflecting the fact that Infinite Descent holds for SG. See also [Berardi and Tatsuta 2019, §4.1] for a detailed discussion of Infinite Descent for 2-hydra.

5 RELATION-BASED CRITERIA

In this section we explore relation-based criteria for Infinite Descent (and Size-Change Principle). We first recall the criterion developed by Lee et al. [2001] and later refined by Ben-Amram and Lee [2007], which can be decided using the Floyd-Warshall-Kleene algorithm for solving the algebraic path problem. We then describe our novel algorithm for deciding this criterion, based on a reduction of the number of paths that need considering.

5.1 Transitive Looping Criterion (Size-Change Principle)

In addition to the Büchi automaton encoding recalled in Section 4.1, Lee et al. [2001] also introduced a more direct criterion based on relation (graph) manipulation to form the composition closure of sloped graphs' relations. We first fix the order $\rightsquigarrow < \downarrow$ on slopes (thinking of $\downarrow$ as being "steeper" than $\rightsquigarrow$), and then define the composition $R \odot Q \subseteq P \times P'' \times \mathsf{S}$ of two sloped relations $R \subseteq P \times P' \times \mathsf{S}$ and $Q \subseteq P' \times P'' \times \mathsf{S}$ as follows: $(R \odot Q)\ (p, p'', s)$ holds just in case s is the largest slope such that there exist $p' \in P'$ and $s_1, s_2 \in \mathsf{S}$ with $R(p, p', s_1)$, $Q(p', p'', s_2)$ and $s = \max(s_1, s_2)$. (There may exist multiple such slopes s_1 and s_2, and the definition chooses a pair that gives the largest value for $\max(s_1, s_2)$.) Thus, sloped relation composition takes the steepest slope available between p and p'' via a route $p \mapsto_R p' \mapsto_Q p''$ for some p'.

We describe (a slightly generalized version of) their criterion in our framework. The key concept for this criterion is the composition closure of a sloped graph.

Definition 5.1. The *composition closure* of a sloped graph $\mathsf{SG} = (\mathsf{V}, \mathsf{E}, \mathsf{Ps}, \mathsf{R})$ is the family of sets of sloped relations $\mathsf{Cl} = (\mathsf{Cl}_{v,v'})_{(v,v') \in \mathsf{V} \times \mathsf{V}}$ defined inductively by:

$$\frac{}{\mathsf{R}_{v,v'} \in \mathsf{Cl}_{v,v'}}\ (v, v') \in \mathsf{E} \qquad \frac{R \in \mathsf{Cl}_{v,v'} \quad Q \in \mathsf{Cl}_{v',v''}}{R \odot Q \in \mathsf{Cl}_{v,v''}}$$

If we want to explicitly indicate the underlying sloped graph SG, we use $\mathsf{Cl}^{\mathsf{SG}}$ instead of Cl. Note that $\mathsf{Cl}_{v,v'} \neq \emptyset$ iff there is a path from v to v'.

The composition closure captures the *position changes along all possible (finite) paths* with the given source and target. Ben-Amram and Lee [2007] show that checking Infinite Descent amounts to checking that, for any idempotent sloped relation connecting a vertex to itself, there exists a position such that the relation describes a strict decrease between that position and itself—in other words, checking that any idempotent vertex-level loop has a position-level loop. We call this property the Idempotent Looping property. In fact, as already observed by Ben-Amram and Lee [2007] and Fogarty and Vardi [2009], this is equivalent to the following property.

Definition 5.2. SG satisfies the *Transitive Looping* property if, for all $v \in \mathsf{V}$ and $P \in \mathsf{Cl}^{\mathsf{SG}}_{v,v}$, P (viewed as a graph) contains a strongly connected component with a $\downarrow$-labeled edge.

The name 'Transitive Looping' derives from the fact that requiring that P contains a strongly connected component with a $\downarrow$-labeled edge is equivalent to requiring that there is a position-level loop in the transitive closure of P.

Proposition 5.3. *The Idempotent Looping property is equivalent to the Transitive Looping property.*

Proof. For a sloped relation R, we denote by $R^{\oplus}$ the *transitive closure* of R. It is straightforward to verify that in a sloped relation R (viewed as a graph), the existence of a strongly connected component with a $\downarrow$-labeled edge is equivalent to the existence of a position p such that $R^{\oplus}(p, p, \downarrow)$.

Now, assume Idempotent Looping. To prove Transitive Looping, let $v \in \mathsf{V}$ and $P \in \mathsf{Cl}^{\mathsf{SG}}_{v,v}$. Then, also $P^{\oplus} \in \mathsf{Cl}^{\mathsf{SG}}_{v,v}$ (because P is finite, and therefore $P^{\oplus} = P^n$ for some n). Since $P^{\oplus}$ is idempotent, by Idempotent Looping we obtain $p \in \mathsf{Ps}(v)$ such that $P^{\oplus}(p, p, \downarrow)$, as desired.

Algorithm 1: Checks Transitive Looping using the Floyd-Warshall-Kleene algorithm.

Input: Sloped Graph SG = (V, E, Ps, R)
Output: True if Transitive Looping holds for SG and False otherwise

```
1  foreach (v, v') ∈ V × V do                                   n² iterations
2      if (v, v') ∈ E then
3          if v = v' and TransLoop(R_{v,v'}) = False then return False    O(w²)
4          Cl_{v,v'} := {R_{v,v'}}
5      else Cl_{v,v'} := ∅
6  foreach v'' ∈ V do                                           n iterations
7      X := (Cl_{v'',v''})*                                     O(w⁴ · 3^{3w²})
8      foreach (v, v') ∈ V × V do                               n² iterations
9          Y_{v,v'} := Cl_{v,v''} ∘ X ∘ Cl_{v'',v'}             O(w⁴ · 3^{2w²})
10         if v = v' then
11             foreach R ∈ Y_{v,v'} do                          O(3^{w²}) iterations
12                 if TransLoop(R) = False then return False    O(w²)
13     foreach (v, v') ∈ V × V do                               O(n²) iterations
14         Cl_{v,v'} := Cl_{v,v'} ∪ Y_{v,v'}                    O(w⁴ · 3^{w²})
15 return True
```

Conversely, assume Transitive Looping. To prove Idempotent Looping, let $v \in \mathsf{V}$ and $P \in \mathsf{Cl}^{\mathsf{SG}}_{v,v}$ such that $P \odot P = P$. By Transitive Looping, we have $p \in \mathsf{Ps}(v)$ such that $P^{\oplus}(p, p, \curvearrowright)$. Since P is idempotent, we have that $P = P^{\oplus}$, and thus $P(p, p, \curvearrowright)$, as desired. □

It may seem that verifying the Transitive Looping property is more computationally expensive than checking the Idempotent Looping property described above since, given an idempotent relation, checking for a position-level loop is in $O(w)$, whereas the condition defining the Transitive Looping property above is in $O(w^2)$ (to compute the strongly connected components). However, this reasoning neglects to consider that we must first perform a check (in $O(w^3)$ time) that a relation is indeed idempotent.

Given that the Idempotent Looping property is equivalent to Infinite Descent [Ben-Amram and Lee 2007], we obtain the following new criterion for Infinite Descent.

Theorem 5.4 (Transitive Looping criterion). *A sloped graph satisfies the Infinite Descent property iff it satisfies the Transitive Looping property.*

The Transitive Looping criterion can be checked by computing the closure, and checking that each relation in the sets $\mathsf{Cl}_{v,v}$ (for each $v \in \mathsf{V}$) satisfies the Transitive Looping property. The composition closure is easily seen to be the solution for the algebraic path problem instantiated to the closed semiring structure on finite sets of sloped relations consisting of: 0 as ∅, 1 as the singleton set having the identity sloped relation as unique element, sum as union, and multiplication as componentwise composition $X \circ Y = \{P \circ Q \mid P \in X \wedge Q \in Y\}$. The standard algorithm for solving the general algebraic path problem is the Floyd-Warshall-Kleene algorithm [Lehmann 1977; Yannakakis 1990], illustrated in Algorithm 1. To check Transitive Looping, we augment the algorithm to make use of a function TransLoop that checks its input sloped relation (viewed as a graph) for a strongly connected component containing a $\curvearrowright$-labeled edge. We have annotated the

algorithm with the complexity of key steps in terms of the parameters n (number of vertices) and w (vertex width) of the sloped graph. The overall complexity is $O(n{\cdot}(w^4{\cdot}3^{3w^2} + n^2{\cdot}w^4{\cdot}3^{2w^2}))$. The execution time is dominated by the **foreach** loop on line 6 (n iterations) and the following two computations nested inside this loop:

- the computation on line 7, namely asteration, which requires $O(w^4 \cdot 3^{3w^2})$ steps;
- the **foreach** loop on line 8 (n^2 iterations) containing the computation of $\mathsf{Cl}_{v,v''} \circ X \circ \mathsf{Cl}_{v'',v'}$ on line 9, which requires $O(w^4 \cdot 3^{2w^2})$ steps (this being the cost of componentwise composition, i.e. the multiplication operation).

Making use of a total comparison function over sloped relations (costing $O(w^2)$ to compare two sloped relations) we can implement the sets of the composition closure using binary search trees, resulting in membership checking and insertion in $O(w^4)$ time (needing $O(\log(3^{w^2})) = O(w^2)$ comparisons). Notice that replacing the $O(w^2)$ Transitive Looping check by the $O(w^3)$ Idempotent Looping check does not affect the overall complexity, since the complexity in both cases is dominated by that of computing the compositions of sloped relations.

Theorem 5.5. *The Transitive Looping criterion is decidable in time* $O(n{\cdot}(w^4{\cdot}3^{3w^2} + n^2{\cdot}w^4{\cdot}3^{2w^2}))$.

We note that other alternatives to exploring the composition closure are possible. For example moving between triples (v, P, v') and $(v, P \odot \mathsf{R}_{v',v''}, v'')$ using breadth-first or iterative deepening depth-first search. However, if we also factor in cycle detection, their worst-case complexity is not better than that of algorithm 1.

This relation-based approach also allows failure of Infinite Descent to be detected 'early' by testing each eligible relation as it is produced, a fact again observed by Ben-Amram and Lee [2007]. Thus, in the case that a sloped graph does not satisfy the Transitive Looping Criterion, this can be determined without necessarily computing the full closure. This contrasts with the automaton-based approaches, which must carry out the same (costly) automaton complementation step regardless of whether Infinite Descent holds or not.

Approximation-based Optimisation. Ben-Amram and Lee [2007, §7.2] also describe an optimisation for this approach based on a notion of approximation between sloped relations. We can extend the ordering $\rightsquigarrow < \searrow$ on slopes to sloped relations pointwise, by first defining a non-strict ordering $P \leq Q$ if and only if $P(p, p', s)$ implies there exists s' such that $Q(p, p', s')$ and $s \leq s'$, for all positions p and p' and slopes s, and then defining the strict ordering on sloped relations by $P < Q$ iff $P \leq Q$ and $P \neq Q$. Note that this only defines a *partial* order on the set of sloped relations. It is easy to see that this relation has the property that $P < Q$ implies $P \odot S < Q \odot S$ and $S \odot P < S \odot Q$.

The optimisation consists in observing that if a sloped relation Q fails the transitive looping check and $P < Q$, then so too does P. This means that only the $<$-minimal elements of the composition closure need to be checked in order to verify Infinite Descent. For a set X of sloped relation, we write $\mathsf{Min}(X)$ to denote the set $\{P \mid P \in X \wedge \nexists Q \in X.\, Q < P\}$ of $<$-minimal elements of X. Note that the following property follows immediately from the definition: $\mathsf{Min}(X) \subseteq X$, for all X.

Definition 5.6. SG satisfies the *Minimal* Transitive Looping property if, for all $v \in \mathsf{V}$ and $P \in \mathsf{Min}(\mathsf{Cl}^{\mathrm{SG}}_{v,v})$, P (viewed as a graph) contains a strongly connected component with a $\searrow$-labeled edge.

Theorem 5.7 (Minimal Transitive Looping criterion). *A sloped graph satisfies the Infinite Descent property iff it satisfies the Minimal Transitive Looping property.*

Proof. By theorem 5.4, it suffices to show that Minimial Transitive Looping is equivalent to Transitive Looping. If a sloped graph satisfies Transitive Looping, then it trivially satisfies Minimal Transitive Looping since $\mathsf{Min}(\mathsf{Cl}_{v,v}) \subseteq \mathsf{Cl}_{v,v}$, for all v. Conversely, if a sloped graph does not satisfy

Transitive Looping, then there is some vertex v and sloped relation $P \in \mathsf{Cl}_{v,v}$ such that P fails the transitive looping check. If $P \in \mathsf{Min}(\mathsf{Cl}_{v,v})$, then the graph does not satisfy Minimal Transitive Looping. Otherwise, there is some $Q \in \mathsf{Min}(\mathsf{Cl}_{v,v})$ such that $Q < P$, and so Q also fails the transitive looping check. Thus, the graph does not satisfy Minimal Transitive Looping in this case either. □

Although using this optimisation can lead to a significant (constant factor) speedup in practice, it does not improve the overall complexity of the approach.

5.2 Order-reduced Transitive Looping Criterion

In fact, as we now show, it is not necessary to compute the full closure $\mathsf{Cl}^{\mathsf{SG}}$ (nor even its approximation-based optimisation) in order to check Infinite Descent. A still smaller 'closure' suffices, which takes advantage of symmetries inherent in the problem domain. For example, supposing (v, v') and (v', v) are edges between distinct vertices in the graph, it is not necessary to consider traces along both $v \cdot v' \cdot v$ and $v' \cdot v \cdot v'$, since the ipath $(v \cdot v')^\omega$ satisfies Infinite Descent if and only if so too does $(v' \cdot v)^\omega$. Our optimised approach relies on fixing (arbitrarily) a total ordering of the vertices in the sloped graph, and observing that any ipath has a tail visiting only vertices 'less than or equal to' the maximum vertex occurring infinitely often. Thus, we do not need to consider paths between vertices v_i and v_j containing vertices strictly greater than $\mathsf{Max}(v_i, v_j)$.

Since the results that we give below hold for arbitrary total orders, we will hereafter assume, for any set V of vertices, an implicit total order $v_1 < \ldots < v_{|\mathsf{V}|}$. In an abuse of vector notation, we will write $\tilde{v}$ to denote this order, and so use $\tilde{v}_i$ to denote the i^{th} vertex of the order, v_i.

Definition 5.8. The *order-reduced composition closure* of a sloped graph $\mathsf{SG} = (\mathsf{V}, \mathsf{E}, \mathsf{Ps}, \mathsf{R})$ is the family of sets of sloped relations $\Omega = (\Omega_{i,j})_{1 \le i,j \le |\mathsf{V}|}$ defined inductively by:

$$\frac{}{\mathsf{R}_{\tilde{v}_i, \tilde{v}_j} \in \Omega_{i,j}} \; (\tilde{v}_i, \tilde{v}_j) \in \mathsf{E} \qquad \frac{P \in \Omega_{i,i} \qquad Q \in \Omega_{i,i}}{P \odot Q \in \Omega_{i,i}} \qquad \frac{P \in \Omega_{i,k} \qquad Q \in \Omega_{k,j}}{P \odot Q \in \Omega_{i,j}} k < \mathsf{Min}(i,j)$$

$$\frac{P \in \Omega_{j,j}}{\mathsf{R}_{\tilde{v}_i, \tilde{v}_j} \odot P \in \Omega_{i,j}} i > j, (\tilde{v}_i, \tilde{v}_j) \in \mathsf{E} \qquad \frac{P_1 \in \Omega_{i,k} \qquad P_2 \in \Omega_{k,j} \qquad P_3 \in \Omega_{j,j}}{P_1 \odot P_2 \odot P_3 \in \Omega_{i,j}} i > j > k$$

As above, we may write Ω^{SG} when we wish to be explicit about the underlying sloped graph SG.

We define analogues of the Transitive Looping property and its Minimal version using the order-reduced closure.

Definition 5.9. SG satisfies the (Minimal) *Order-reduced Transitive Looping* property if, for all $i \le |\mathsf{V}|$ and $P \in \Omega^{\mathsf{SG}}_{i,i}$ (resp. $\mathsf{Min}(\Omega^{\mathsf{SG}}_{i,i})$), P (viewed as a graph) contains a strongly connected component with a $\hookrightarrow$-labeled edge.

We now consider the equivalence of the Order-reduced Transitive Looping property to the Infinite Descent property. One direction follows as a corollary of the fact that the order-reduced closure is a subset of the full composition closure.

Lemma 5.10. *For a sloped graph* $\mathsf{SG} = (\mathsf{V}, \mathsf{E}, \mathsf{Ps}, \mathsf{R})$, $\Omega^{\mathsf{SG}}_{i,j} \subseteq \mathsf{Cl}^{\mathsf{SG}}_{\tilde{v}_i, \tilde{v}_j}$ *for all* $i, j \le |\mathsf{V}|$.

Proof. By a straightforward induction on the definition of Ω. □

The converse direction obtains, essentially, because the sets $\Omega_{i,i}$ capture the sloped relations corresponding to just enough finite loops for considering all possible ipaths. To show this, we first consider some different categories of (finite) paths. In the following, we fix a sloped graph $\mathsf{SG} = (\mathsf{V}, \mathsf{E}, \mathsf{Ps}, \mathsf{R})$.

Definition 5.11. We define a number of (natural number indexed) families of strings, or sequences, of vertices using the following grammar.[2]

$$\begin{array}{rll} \text{loops} & \lambda(i) ::= \tilde{v}_i\ \rho(i)\ \tilde{v}_i & \\ \text{routes} & \rho(i) ::= [\rho(j)]\ \big(\ \tilde{v}_j \mid \sigma(j)\ \big)\ [\rho(j)] & (\text{for all } j < i) \\ \text{circuits} & \sigma(i) ::= \tilde{v}_i\ [\tau(i)]\ \tilde{v}_i & \\ \text{trails} & \tau(i) ::= \tilde{v}_j \mid \tilde{v}_j\ \tau(i) & (\text{for all } j \leq i) \end{array}$$

For any string φ of symbols (i.e. vertices or non-terminals), we denote by $\mathsf{Lang}(\varphi)$ the *language* of φ—i.e. the set of sentences, or terminal strings, derivable from φ.

Notice that $\mathsf{Lang}(\tau(i))$ contains all sequences of vertices less than or equal to $\tilde{v}_i$. Thus, $\mathsf{Lang}(\sigma(i))$ contains sequences of vertices starting and ending at $\tilde{v}_i$ that consist of vertices less than or equal to $\tilde{v}_i$. The definitions above are carefully formulated to yield the following properties about (arbitrary) sequences of vertices (i.e. not only sequences that are actual paths in the graph).

Lemma 5.12. *The following hold:*

(1) $(v_k)_{k<n} \in \mathsf{Lang}(\rho(i))$ *if* $v_k < \tilde{v}_i$ *for all* $k < n$.
(2) $(v_k)_{k<n} \in \mathsf{Lang}(\lambda(i))$ *if* $n > 2$, $v_0 = v_{n-1} = \tilde{v}_i$, *and* $v_k < v_0$ *for all* $0 < k < n-1$.

Proof. (1) By strong induction on i. Let $\tilde{v}_j$ be the maximum vertex appearing in $(v_k)_{k<n}$. Since $v_k < \tilde{v}_i$ for all $k < n$, we have that $j < i$. Let $0 \leq m, m' < n$ be such that v_m and $v_{m'}$ are the first and last occurrences, respectively, of $\tilde{v}_j$ in $(v_k)_{k<n}$. Thus, we have $(v_k)_{m\leq k\leq m'} \in \mathsf{Lang}(\tilde{v}_j \mid \sigma(j))$. Moreover, when $m > 0$, it must be that $v_k < \tilde{v}_j$ for all $k < m$ and so, by the inductive hypothesis, $(v_k)_{k<m} \in \mathsf{Lang}(\rho(j))$. Similarly, when $m' < n-1$, it must be that $v_k < \tilde{v}_j$ for all $m' \leq k < n$ and so, by the inductive hypothesis, $(v_k)_{m'\leq k<n} \in \mathsf{Lang}(\rho(j))$. Therefore, it follows that $(v_j)_{j<n} \in \mathsf{Lang}([\rho(j)]\ (\ \tilde{v}_j \mid \sigma(j)\ )\ [\rho(j)])$. So, by definition 5.11, $(v_j)_{j<n} \in \mathsf{Lang}(\rho(i))$, since $j < i$.
(2) Immediate, by definition 5.11 and part (1). □

Given a non-trivial path $\pi = (v_i)_{i<n}$, we denote by $\sum \pi$ the composition of the sloped relations along the edges of π; i.e. $\sum \pi = \mathsf{R}_{v_0,v_1} \odot \ldots \odot \mathsf{R}_{v_{n-2},v_{n-1}}$. Notice that for a path $\pi = \pi_1 \cdot \pi_2$, with π_1 and π_2 both non-trivial, we have that $\sum \pi = (\sum \pi_1) \odot (\sum \pi_2)$. The following result is the key property of the order-reduced composition closure.

Lemma 5.13. *For all* i, j *with* $1 \leq i, j \leq |\mathsf{V}|$, *and all paths* π, *the following hold:*

(i) if $i < j$ *and* $\pi \in \mathsf{Lang}(\tilde{v}_i\ [\rho(i)]\ \tilde{v}_j)$, *then* $\sum \pi \in \Omega_{i,j}$;
(ii) if $i > j$ *and* $\pi \in \mathsf{Lang}(\tilde{v}_i\ [\rho(j)]\ \tilde{v}_j)$ *or* $\pi \in \mathsf{Lang}(\tilde{v}_i\ [\rho(j)]\ \sigma(j))$, *then* $\sum \pi \in \Omega_{i,j}$;
(iii) if $i = j$ *and* $\pi \in \mathsf{Lang}(\sigma(i))$ *then* $\sum \pi \in \Omega_{i,j}$.

Proof. By wellfounded induction on the lexicographic ordering on $\mathbb{N} \times \mathbb{N}$.

(i): We distinguish the following cases.
- ($\pi \in \mathsf{Lang}(\tilde{v}_i\ \tilde{v}_j)$): Then $\pi = \tilde{v}_i \cdot \tilde{v}_j$. Therefore $(\tilde{v}_i, \tilde{v}_j) \in \mathsf{E}$ and $\sum \pi = \mathsf{R}_{\tilde{v}_i,\tilde{v}_j}$. Thus the result follows from the base case in definition 5.8.
- ($\pi \in \mathsf{Lang}(\tilde{v}_i\ [\rho(k)]\ (\ \tilde{v}_k \mid \sigma(k)\ )\ [\rho(k)]\ \tilde{v}_j)$, for some $k < i$): Then $\pi = \pi' \cdot \pi''$, with $\pi' \in \mathsf{Lang}(\tilde{v}_i\ [\rho(k)]\ (\ \tilde{v}_k \mid \sigma(k)\ ))$ and $\pi'' \in \mathsf{Lang}(\tilde{v}_k\ [\rho(k)]\ \tilde{v}_j)$. Since $k < i < j$, it follows that $(i,k) < (i,j)$ and so, by the part (ii) of the inductive hypothesis, we have that $\sum \pi' \in \Omega_{i,k}$. Furthermore, since $k < i$, it follows that $(k,j) < (i,j)$ and so, by part (i) of the inductive hypothesis, we have that $\sum \pi'' \in \Omega_{k,j}$. Therefore, since $k < i = \mathsf{Min}(i,j)$, it follows that $\sum \pi = (\sum \pi') \odot (\sum \pi'') \in \Omega_{i,j}$ by definition 5.8.

[2]Following EBNF syntax, the notation $[\ldots]$ indicates optional occurrence of a string of symbols.

(ii): We distinguish the following cases.
- ($\pi \in \mathsf{Lang}(\tilde{\nu}_i\ \tilde{\nu}_j)$): Then $\pi = \tilde{\nu}_i \cdot \tilde{\nu}_j$. Therefore $(\tilde{\nu}_i, \tilde{\nu}_j) \in \mathsf{E}$ and $\sum \pi = \mathsf{R}_{\tilde{\nu}_i,\tilde{\nu}_j}$. Thus the result follows from the base case in definition 5.8.
- ($\pi \in \mathsf{Lang}(\tilde{\nu}_i\ \sigma(j))$): Then $\pi = \tilde{\nu}_i \cdot \pi'$, with $(\tilde{\nu}_i, \tilde{\nu}_j) \in \mathsf{E}$ and $\pi' \in \mathsf{Lang}(\sigma(j))$. Since $j < i$, it follows that $(j, j) < (i, j)$ and so, by part (iii) of the inductive hypothesis, we have that $\sum \pi' \in \Omega_{j,j}$. Therefore, it follows that $\sum \pi = \mathsf{R}_{i,j} \odot (\sum \pi') \in \Omega_{i,j}$ by definition 5.8.
- ($\pi \in \mathsf{Lang}(\tilde{\nu}_i\ [\rho(k)]\ (\ \tilde{\nu}_k \mid \sigma(k)\ )\ [\rho(k)]\ \tilde{\nu}_j)$, for some $k < j$): Then $\pi = \pi' \cdot \pi''$, with $\pi' \in \mathsf{Lang}(\tilde{\nu}_i\ [\rho(k)]\ (\ \tilde{\nu}_k \mid \sigma(k)\ ))$ and $\pi'' \in \mathsf{Lang}(\tilde{\nu}_k\ [\rho(k)]\ \tilde{\nu}_j)$. Since $k < j$, it follows that $(i, k) < (i, j)$ and so, by the part (ii) of the inductive hypothesis, we have that $\sum \pi' \in \Omega_{i,k}$. Furthermore, since $k < j < i$, it follows that $(k, j) < (i, j)$ and so, by part (i) of the inductive hypothesis, we have that $\sum \pi'' \in \Omega_{k,j}$. Therefore, since $k < j = \mathsf{Min}(i, j)$, it follows that $\sum \pi = (\sum \pi') \odot (\sum \pi'') \in \Omega_{i,j}$ by definition 5.8.
- ($\pi \in \mathsf{Lang}(\tilde{\nu}_i\ [\rho(k)]\ (\ \tilde{\nu}_k \mid \sigma(k)\ )\ [\rho(k)]\ \sigma(j))$, for some $k < j$): Then $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$, with $\pi_1 \in \mathsf{Lang}(\tilde{\nu}_i\ [\rho(k)]\ (\ \tilde{\nu}_k \mid \sigma(k)\ ))$, $\pi_2 \in \mathsf{Lang}(\tilde{\nu}_k\ [\rho(k)]\ \tilde{\nu}_j)$, and $\pi_3 \in \mathsf{Lang}(\sigma(j))$. Since $k < j$, it follows that $(i, k) < (i, j)$ and so, by part (ii) of the inductive hypothesis, we have that $\sum \pi_1 \in \Omega_{i,k}$. Since, moreover, that $j < i$, it follows that $(k, j) < (i, j)$ and so, by part (i) of the inductive hypothesis, we have that $\sum \pi_2 \in \Omega_{k,j}$. Finally, since $j < i$, it follows that $(j, j) < (i, j)$ and so, by part (iii) of the inductive hypothesis, we have that $\sum \pi_3 \in \Omega_{j,j}$. Therefore, since $k < j < i$, it follows that $\sum \pi = (\sum \pi_1) \odot (\sum \pi_2) \odot (\sum \pi_3) \in \Omega_{i,j}$, by definition 5.8.

(iii) Then π starts and ends with $\tilde{\nu}_i$ and contains only vertices less than or equal than $\tilde{\nu}_i$. We proceed by induction on the number of occurrences of $\tilde{\nu}_i$. In the base case, π contains only two occurrences: the first and last vertices. We distinguish two subcases.
1. Then $\pi = \tilde{\nu}_i \cdot \tilde{\nu}_i$. Therefore $(\tilde{\nu}_i, \tilde{\nu}_i) \in \mathsf{E}$ and $\sum \pi = \mathsf{R}_{\tilde{\nu}_i,\tilde{\nu}_j}$. Thus the result follows from the base case in definition 5.8.
2. Then $\pi = (\nu_j)_{j<n}$ such that $\nu_0 = \nu_{n-1} = \tilde{\nu}_i$ and $\nu_k < \tilde{\nu}_i$ for all $0 < k < n - 1$. Thus, by lemma 5.12(2), $\pi \in \mathsf{Lang}(\lambda(i))$ and so $\pi \in \mathsf{Lang}(\tilde{\nu}_i\ [\rho(k)]\ (\ \tilde{\nu}_k \mid \sigma(k)\ )\ [\rho(k)]\ \tilde{\nu}_i)$ for some $k < i$, by definition. It then follows that $\pi = \pi' \cdot \pi''$, with either $\pi' \in \mathsf{Lang}(\tilde{\nu}_i\ [\rho(k)]\ \tilde{\nu}_k)$ or $\pi' \in \mathsf{Lang}(\tilde{\nu}_i\ [\rho(k)]\ \sigma(k))$, and $\pi'' \in \mathsf{Lang}(\tilde{\nu}_k\ [\rho(k)]\ \tilde{\nu}_i)$. Since $k < i$, it also follows that $(i, k) < (i, i)$ and so, by part (ii) of the outer induction, we have that $\sum \pi' \in \Omega_{i,k}$. Similarly, $(k, i) < (i, i)$ and so, by part (i) of the outer induction, we have that $\sum \pi'' \in \Omega_{k,i}$. Therefore, it follows that $\sum \pi = (\sum \pi') \odot (\sum \pi'') \in \Omega_{i,i}$.

In the inductive step, we can decompose π as $\pi = \pi_1 \cdot \pi_2$ with $\pi_1, \pi_2 \in \mathsf{Lang}(\sigma(i))$ by picking an intermediate occurrence of $\tilde{\nu}_i$. Thus, both π_1 and π_2 have one less occurrence each of $\tilde{\nu}_i$ than π. So, by the inner induction, we have that $\sum \pi_1 \in \Omega_{i,i}$ and $\sum \pi_2 \in \Omega_{i,i}$. Therefore, it follows that $\sum \pi = (\sum \pi_1) \odot (\sum \pi_1) \in \Omega_{i,i}$ by definition 5.8. □

The following Ramsey-theoretic property is what allows us to link the Order-reduced Transitive Looping property with Infinite Descent.

Proposition 5.14. *Suppose* SG *does not satisfy Infinite Descent, then there is a path π in* SG *such that the ipath $(\pi)^\omega$ is not descending.*

Theorem 5.15 (Order-reduced Transitive Looping criterion). *A sloped graph satisfies the Infinite Descent property iff it satisfies the (Minimal) Order-reduced Transitive Looping property.*

Proof. Order-reduced Transitive Looping and its Minimal version are equivalent by similar reasoning to that employed in the proof of theorem 5.7. The equivalence of Infinite Descent and Order-reduced Transitive Looping can be demonstrated as follows. The 'only if' direction is a straightforward corollary of theorem 5.4 and lemma 5.10. For the 'if' direction, we prove the

contrapositive. Suppose the Infinite Descent property fails, then by proposition 5.14 there is some path $\pi = (v_j)_{j<n}$ in the graph such that the ipath $(\pi)^\omega$ is not infinitely descending. Let $k < n$ and i be such that $v_k = \tilde{v}_i$ is a maximum node occurring in π. Then, the ipath $((v_j)_{k\leq j<n}\,(v_j)_{j<k})^\omega$ is not infinitely descending either. Now, notice that the path $\pi' = v_k \ldots v_{n-1}\, v_0 \ldots v_k \in \mathsf{Lang}(\sigma(i))$. We must have that the sloped relation $P = \sum \pi'$ fails to satisfy the Transitive Looping property, and thus that the Order-reduced Transitive Looping property fails since, by lemma 5.13(iii), $P \in \Omega_{i,i}$. To see that P fails to satisfy the Transitive Looping property, suppose otherwise and let $m \in \mathbb{N}$ be the least m such that P^m is the transitive closure of P. Then there is a trace $(p_j)_{j<n*m}$ for $((v_j)_{k\leq j<n}\,(v_j)_{j<k})^m$ containing a $\searrow$ slope, which entails the contradiction that the ipath $((v_j)_{k\leq j<n}\,(v_j)_{j<k})^\omega$ is descending. □

It is immediate from definition 5.8 that the order-reduced composition closure can be computed using dynamic programming, since each set $\Omega_{i,j}$ depends only sets $\Omega_{i',j'}$ where $(i', j') <_{\text{LEX}} (i, j)$ (and the sloped relations in the input graph). A fixpoint computation is only necessary in the case $i = j$, and can operate over an initial set of relations obtained directly from the preceding sets and the input graph.

Lemma 5.16. *$P \in \Omega_{i,i}$ if and only if $P = Q_1 \odot \ldots \odot Q_n$ (for some $n > 0$) where, for each $0 < j \leq n$, either $Q_j = \mathsf{R}_{\tilde{v}_i,\tilde{v}_i}$ or $Q_j = Q' \odot Q''$ for some $Q' \in \Omega_{i,k}$ and $Q'' \in \Omega_{k,i}$, with $k < i$.*

Proof. Straightforward, by induction on n for the 'if' direction, and by induction on the definition of Ω for the 'only if' direction. □

It is less obvious that the minimal sets of the order-reduced closure can be similarly computed (i.e. without first computing the full order-reduced closure, and then filtering the $<$-minimal elements). However, the following results show that the sets $\mathsf{Min}(\Omega_{i,j})$ do indeed depend only on the sets $\mathsf{Min}(\Omega_{i',j'})$ with $(i', j') <_{\text{LEX}} (i, j)$ (and the sloped relations in the input graph that also appear within the minimal order-reduced closure sets).

Lemma 5.17. *The following hold:*

(1) If $P \in \mathsf{Min}(\Omega_{i,j})$ with $i < j$, then either $P = \mathsf{R}_{\tilde{v}_i,\tilde{v}_j}$ or $P = Q_1 \odot Q_2$ for some $k < i$ such that $Q_1 \in \mathsf{Min}(\Omega_{i,k})$ and $Q_2 \in \mathsf{Min}(\Omega_{k,j})$.

(2) If $P \in \mathsf{Min}(\Omega_{i,j})$ with $j < i$, then one of the following holds.

(a) $P = \mathsf{R}_{\tilde{v}_i,\tilde{v}_j}$.

(b) $P = Q_1 \odot Q_2$ for some $k < i$ such that $Q_1 \in \mathsf{Min}(\Omega_{i,k})$ and $Q_2 \in \mathsf{Min}(\Omega_{k,j})$.

(c) $P = \mathsf{R}_{\tilde{v}_i,\tilde{v}_j} \odot Q$, with $Q \in \mathsf{Min}(\Omega_{j,j})$; moreover, there are no $Q_1 \in \mathsf{Min}(\Omega_{i,k}), Q_2 \in \mathsf{Min}(\Omega_{k,j})$ such that $Q_1 \odot Q_2 < \mathsf{R}_{\tilde{v}_i,\tilde{v}_j}$.

(d) $P = Q_1 \odot Q_2 \odot Q_3$ with $Q_1 \in \mathsf{Min}(\Omega_{i,k})$, $Q_2 \in \mathsf{Min}(\Omega_{k,j})$, and $Q_3 \in \mathsf{Min}(\Omega_{j,j})$ for some $k < j$; moreover, there are no $Q_1' \in \mathsf{Min}(\Omega_{i,k}), Q_2' \in \mathsf{Min}(\Omega_{k,j})$ such that $Q_1' \odot Q_2' < Q_1 \odot Q_2$.

(3) If $P \in \mathsf{Min}(\Omega_{i,i})$, then $P = Q_1 \odot \ldots \odot Q_n$ (for some $n > 0$) and for each $0 < j \leq n$:

(i) either $Q_j = \mathsf{R}_{\tilde{v}_i,\tilde{v}_i}$ or $Q_j = Q' \odot Q''$ for some $Q' \in \mathsf{Min}(\Omega_{i,k})$, $Q'' \in \mathsf{Min}(\Omega_{k,i})$, $k < i$; and

(ii) there are no $S \in \mathsf{Min}(\Omega_{i,k})$ and $S' \in \mathsf{Min}(\Omega_{k,j})$ with $k < i$ such that $S \odot S' < Q_j$.

Proof. (1): Suppose $P \in \mathsf{Min}(\Omega_{i,j})$, with $i < j$. Then $P \in \Omega_{i,j}$, since $\mathsf{Min}(\Omega_{i,j}) \subseteq \Omega_{i,j}$, and so by definition 5.8 that there are two cases to consider. If $P = \mathsf{R}_{\tilde{v}_i,\tilde{v}_j}$, then the result follows immediately. Otherwise, $P = Q_1 \odot Q_2$ with $Q_1 \in \Omega_{i,k}$ and $Q_2 \in \Omega_{k,j}$ for some $k < i$. We must have that $Q_1 \in \mathsf{Min}(\Omega_{i,k})$ and $Q_2 \in \mathsf{Min}(\Omega_{k,j})$ since, supposing otherwise we derive a contradiction. If $Q_1 \notin \mathsf{Min}(\Omega_{i,k})$ then there is $Q_1' \in \mathsf{Min}(\Omega_{i,k})$ with $Q_1' < Q_1$. Then it follows that also $Q_1' \in \Omega_{i,k}$ and so, from definition 5.8, that $Q_1' \odot Q_2 \in \Omega_{i,j}$ with $Q_1' \odot Q_2 < Q_1 \odot Q_2$. But then $P = Q_1 \odot Q_2 \notin \mathsf{Min}(\Omega_{i,j})$, contradicting the original assumption. The case that $Q_2 \notin \mathsf{Min}(\Omega_{k,j})$ follows symmetrically.

(2): Suppose $P \in \mathsf{Min}(\Omega_{i,j})$, with $j < i$. Then $P \in \Omega_{i,j}$, since $\mathsf{Min}(\Omega_{i,j}) \subseteq \Omega_{i,j}$, and so by definition 5.8 that there are four cases to consider.

(a) If $P = \mathsf{R}_{\tilde{v}_i,\tilde{v}_j}$, then the result follows immediately.

(b) If $P = Q_1 \odot Q_2$ with $Q_1 \in \Omega_{i,k}$ and $Q_2 \in \Omega_{k,j}$ for some $k < j$, then we must have that $Q_1 \in \mathsf{Min}(\Omega_{i,k})$ and $\mathsf{Min}(Q_2 \in \Omega_{k,j})$, by similar reasoning as that demonstrated in part (1).

(c) If $P = \mathsf{R}_{\tilde{v}_i,\tilde{v}_j} \odot Q$, with $Q \in \Omega_{j,j}$, then we must have $Q \in \mathsf{Min}(\Omega_{j,j})$ since supposing otherwise we derive a contradiction. If $Q \notin \mathsf{Min}(\Omega_{j,j})$ then, by definition, there is some $Q' \in \mathsf{Min}(\Omega_{j,j})$ such that $Q' < Q$. Then, $Q' \in \Omega_{j,j}$ and so by definition 5.8, we have $\mathsf{R}_{\tilde{v}_i,\tilde{v}_j} \odot Q' \in \Omega_{i,j}$. But we also have that $\mathsf{R}_{\tilde{v}_i,\tilde{v}_j} \odot Q' < \mathsf{R}_{\tilde{v}_i,\tilde{v}_j} \odot Q = P$, whence it follows that $P \notin \mathsf{Min}(\Omega_{i,j})$. Moreover, there are no $Q_1 \in \mathsf{Min}(\Omega_{i,k}), Q_2 \in \mathsf{Min}(\Omega_{k,j})$ such that $Q_1 \odot Q_2 < \mathsf{R}_{\tilde{v}_i,\tilde{v}_j}$ since supposing otherwise we have by definition 5.8 that $Q_1 \odot Q_2 \odot Q \in \Omega_{i,j}$ and, furthermore, that $Q_1 \odot Q_2 \odot Q < \mathsf{R}_{\tilde{v}_i,\tilde{v}_j} \odot Q = P$, from which it would follow that $P \notin \mathsf{Min}(\Omega_{i,j})$.

(d) If $P = Q_1 \odot Q_2 \odot Q_3$ with $Q_1 \in \Omega_{i,k}$, $Q_2 \in \Omega_{k,j}$ and $Q_3 \in \Omega_{j,j}$, for some $k < j$, then we must have that $Q_1 \in \mathsf{Min}(\Omega_{i,k})$, $\mathsf{Min}(Q_2 \in \Omega_{k,j})$, and $\mathsf{Min}(Q_3 \in \Omega_{j,j})$ since supposing otherwise we derive a contradiction. If $Q_1 \notin \mathsf{Min}(\Omega_{i,k})$ then, by definition, there is some $Q' \in \mathsf{Min}(\Omega_{i,k})$ such that $Q' < Q_1$. Then, by definition 5.8, we have that $Q' \odot Q_2 \odot Q_3 \in \Omega_{i,j}$ and also that $Q' \odot Q_2 \odot Q_3 < Q_1 \odot Q_2 \odot Q_3 = P$, in which case it follows that $P \notin \mathsf{Min}(\Omega_{i,j})$. The cases that $Q_2 \notin \mathsf{Min}(\Omega_{k,j})$ and $Q_3 \notin \mathsf{Min}(\Omega_{j,j})$ are symmetric. Moreover, there are no $Q_1' \in \mathsf{Min}(\Omega_{i,k}), Q_2' \in \mathsf{Min}(\Omega_{k,j})$ such that $Q_1' \odot Q_2' < Q_1 \odot Q_2$, since supposing otherwise we have by definition 5.8 that $Q_1' \odot Q_2' \odot Q_3 \in \Omega_{i,j}$ and, furthermore that $Q_1' \odot Q_2' \odot Q_3 < Q_1 \odot Q_2 \odot Q_3 = P$, from which it would follow that $P \notin \mathsf{Min}(\Omega_{i,j})$.

(3): Suppose $P \in \mathsf{Min}(\Omega_{i,i})$. Then $P \in \Omega_{i,i}$, since $\mathsf{Min}(\Omega_{i,i}) \subseteq \Omega_{i,i}$, and so by lemma 5.16 we have that $P = Q_1 \odot \ldots \odot Q_n$ (for some $n > 0$) such that, for each $0 < j \leq n$, either $Q_j = \mathsf{R}_{\tilde{v}_i,\tilde{v}_j}$ or $Q_j = Q' \odot Q''$ for some $Q' \in \Omega_{i,k}$ and $Q'' \in \Omega_{k,i}$, with $k < i$. A straightforward induction on n yields that then there exist $Q_1', \ldots, Q_n'$ such that $Q_1' \odot \ldots \odot Q_n' \leq Q_1 \odot \ldots \odot Q_n$ and, for each $0 < j \leq n$, either $Q_j' = \mathsf{R}_{\tilde{v}_i,\tilde{v}_j}$ or $Q_j' = S \odot S'$ for some $S \in \mathsf{Min}(\Omega_{i,k})$ and $S' \in \mathsf{Min}(\Omega_{k,i})$, with $k < i$. Notice that, by lemma 5.16, $Q_1' \odot \ldots \odot Q_n' \in \Omega_{i,i}$ and so we must have that $P = Q_1' \odot \ldots \odot Q_n'$, since otherwise we would have that $Q_1' \odot \ldots \odot Q_n' < P$, from which it would follow that $P \notin \mathsf{Min}(\Omega_{i,i})$. Moreover, for an arbitrary Q_j', there can be no $R \in \mathsf{Min}(\Omega_{i,k})$ and $R' \in \mathsf{Min}(\Omega_{k,i})$ with $k < i$ such that $R \odot R' < Q_j'$, since otherwise, by lemma 5.16, we would have that $P' = Q_1' \odot \ldots \odot Q_{j-1}' \odot (R \odot R') \odot Q_{j+1}' \odot \ldots \odot Q_n' \in \Omega_{i,i}$ and, furthermore, that $P' < Q_1' \odot \ldots \odot Q_n' = P$, from which it would follow that $P \notin \mathsf{Min}(\Omega_{i,i})$. □

Algorithm 2 checks the Minimal Order-reduced Transitive Looping Property for a sloped graph, by computing the minimal order-reduced composition closure and checking the transitive looping property for each relation added to the sets $\mathsf{Min}(\Omega_{i,i})$ ($i \leq |\mathsf{V}|$). The correctness of the algorithm follows from lemma 5.17. As for the approach using the full composition closure, it may also 'fail fast', avoiding to compute the entire minimal order-reduced closure in the case that the sloped graph does not satisfy Infinite Descent.

The overall complexity of the algorithm is $O(n^3 \cdot w^2 \cdot 3^{3w^2})$. The basic structure employs a nested loop (lines 1 and 2) to compute each individual set in the order-reduced closure. In computing the closure sets, the algorithm maintains the invariant that they contain only the minimal relations processed so far; thus execution results in the minimal order-reduced closure. The main operation for this is to update the current set with a new sloped relation R if it has no predecessor, and in this case to also remove relations that it precedes. This happens on Lines 14, 21 and 31. The simplest[3] way of doing this is by iterating through the current set: Relations preceded by R are discarded along

[3] A specialised data structure for storing partially ordered sets is described by Daskalakis et al. [2011]. Using this would be optimal, but the complexity is anyway dependent on the width of the partial order, which in our case we believe is proportional to 3^{w^2} (i.e. the maximum number of sloped relations).

Algorithm 2: Checks Minimal Order-reduced Transitive Looping.

Input: Sloped Graph SG = (V, E, Ps, R)
Requires: Fixed order $\tilde{v}$ on V
Output: True if Minimal Order-reduced Transitive Looping holds for SG, False otherwise

```
1  for i := 1 to |V| do                                                    n iterations
2      for j := 1 to |V| do                                                n iterations
3          if (ṽ_i, ṽ_j) ∈ E then
4              if i = j and TransLoop(R_{ṽ_i,ṽ_j}) = False then return False   O(w^2)
5              Ω_{i,j} := {R_{ṽ_i,ṽ_j}}
6          else Ω_{i,j} := ∅
7          Visited := Ω_{i,j}
8          for k := 1 to Min(i, j) − 1 do                                  O(n) iterations
9              foreach (P, Q) ∈ Ω_{i,k} × Ω_{k,j} do                       O(3^{2w^2}) iterations
10                 R := P ⊙ Q                                              O(w^3)
11                 if R ∉ Visited then                                     O(w^4)
12                     if i = j and TransLoop(R) = False then return False O(w^2)
13                     Visited := Visited ∪ {R}                            O(w^4)
14                     Ω_{i,j} := Min(Ω_{i,j} ∪ {R})                       O(3^{w^2} · w^2)
15         X := Ω_{i,j}                                                    O(3^{w^2})
16         if j < i then
17             foreach (P, Q) ∈ X × Ω_{j,j} do                             O(3^{2w^2}) iterations
18                 R := P ⊙ Q                                              O(w^3)
19                 if R ∉ Visited then                                     O(w^4)
20                     Visited := Visited ∪ {R}                            O(w^4)
21                     Ω_{i,j} := Min(Ω_{i,j} ∪ {R})                       O(3^{w^2} · w^2)
22         else if i = j then
23             q := Init-Queue(X)                                          O(3^{w^2})
24             while q not empty do                                        O(3^{w^2}) iterations
25                 Q := Dequeue(q)
26                 foreach P ∈ X do                                        O(3^{w^2}) iterations
27                     R := P ⊙ Q                                          O(w^3)
28                     if R ∉ Visited then                                 O(w^4)
29                         if TransLoop(R) = False then return False       O(w^2)
30                         Visited := Visited ∪ {R}                        O(w^4)
31                         Ω_{i,i} := Min(Ω_{i,i} ∪ {R})                   O(3^{w^2} · w^2)
32                         Enqueue(q, R)
33 return True
```

the way, and R is discarded on encountering a relation preceding it, or else added if no such element is found. For this, it suffices to implement the sets $\Omega_{i,j}$ as linked lists, and the cost of this operation is $O(3^{w^2} \cdot w^2)$—iterating over $O(3^{w^2})$ elements, performing a $O(w^2)$ comparison each time. In order to avoid carrying out this costly update operation more than necessary, the algorithm maintains a collection, Visited, of the sloped relations that have been processed within a given iteration of the nested loop on lines 1 and 2, for which we only need to be able to check membership and insert new elements. As in algorithm 1, this can be implemented using a binary search tree, making these operations $O(w^4)$. The overall cost of computing and processing each new sloped relation is thus $O(3^{w^2} \cdot w^2)$, since the complexity of updating and minimising the closure set dominates the other operations (composing relations, checking membership within and updating the collection of visited relations, and checking the transitive looping property). Given that the closure sets are implemented as linked lists, the copy and queue initialisation operations (on lines 15 and 23, respectively) have $O(3^{w^2})$ complexity. Enqueue and dequeue operations are, of course, $O(1)$.

Counter-intuitively, the worst-case complexity of computing the full (i.e. not minimal) order-reduced closure is slightly lower. If we do not need to iterate over the closure sets to check precedence between sloped relations, we simply need to add new sloped relations and check for membership; thus the collection, Visited, suffices. Then, the cost of computing and processing sloped relations is only $O(w^4)$, and so the overall complexity of the algorithm is reduced to $O(n^3 \cdot w^4 \cdot 3^{2w^2})$. However, as demonstrated by our experimental results, the minimal order-reduced closure often works better in practice because it can significantly reduce the number of iterations needed for the loops on lines 9, 17, 24 and 26.

We therefore obtain the following complexity result:

Theorem 5.18. *The Order-reduced Transitive Looping criterion is decidable in time* $O(n^3 \cdot w^4 \cdot 3^{2w^2})$.

It is interesting to compare this to our complexity result of $O(n{\cdot}(w^4{\cdot}3^{3w^2} + n^2{\cdot}w^4{\cdot}3^{2w^2})) = O(n{\cdot}w^4{\cdot}3^{3w^2} + n^3{\cdot}w^4{\cdot}3^{2w^2})$ for the Transitive Looping criterion. This quite clearly gives a measure, namely $O(n{\cdot}w^4{\cdot}3^{3w^2})$, of the saving obtained by avoiding checking "duplicate" ipaths, e.g. both $(v \cdot v')^\omega$ and $(v' \cdot v)^\omega$, for Infinite Descent.

Example 5.19. We now consider a family of sloped graphs showing the (potentially large) difference between the full and order-reduced composition closures. For all $n > 0$, take $\mathsf{SG}_n = (\mathsf{V}, \mathsf{E}, \mathsf{Ps}, \mathsf{R})$ defined by: $\mathsf{V} = \{v_1, \ldots, v_n\}$, $\mathsf{E} = \{(v_1, v_n), (v_n, v_1), \ldots, (v_{n-1}, v_n), (v_n, v_{n-1}), (v_n, v_n)\}$, $\mathsf{Ps}(v_1) = \ldots = \mathsf{Ps}(v_n) = \{p_1, \ldots, p_n\}$, and the following sloped relations:

- $\mathsf{R}_{v_n,v_j} = \{(p_k, p_k, \rightsquigarrow) \mid 1 \leq k \leq n\}$ for all $j < n$.
- $\mathsf{R}_{v_j,v_n} = \{(p_j, p_j, \searrow)\} \cup \{(p_k, p_k, \rightsquigarrow) \mid k \neq j \wedge 1 \leq k \leq n\}$ for all $j < n$.
- $\mathsf{R}_{v_n,v_n} = \{(p_n, p_n, \searrow)\} \cup \{(p_k, p_k, \rightsquigarrow) \mid 1 \leq k < n\}$.

This is a graph with n vertices, and n simple loops all having vertex v_n in common. The graph is strongly connected, so we can visit every vertex from every other one, but all connections *must* go through the vertex v_n. Because each loop decreases a different position, we can generate (by composition) sloped relations witnessing every possible subset of decreasing positions by taking paths traversing different combinations of loops. Even for $n = 3$, we can see a significant difference between the sizes of the (minimal) full and order-reduced composition closures. Writing P_X, where $X \subseteq \{1, 2, 3\}$, for the sloped relation defined by $P_X = \{(p_k, p_k, \searrow) \mid k \in X\} \cup \{(p_k, p_k, \rightsquigarrow) \mid 1 \leq k \leq 3 \wedge k \notin X\}$, the full composition closure is computed as follows, considering each iteration of the outermost loop on line 6 of algorithm 1:

(Iteration 1): $\mathsf{Cl}_{v_1,v_1} = \mathsf{Cl}_{v_1,v_2} = \mathsf{Cl}_{v_2,v_1} = \mathsf{Cl}_{v_2,v_2} = \varnothing$ $\quad \mathsf{Cl}_{v_1,v_3} = \{\, P_{\{1\}} \,\}$ $\quad \mathsf{Cl}_{v_2,v_3} = \{\, P_{\{2\}} \,\}$

$\mathsf{Cl}_{v_3,v_1} = \mathsf{Cl}_{v_3,v_2} = \{\, P_{\varnothing} \,\}$ $\quad \mathsf{Cl}_{v_3,v_3} = \mathsf{Cl}_{v_3,v_3} = \{\, P_{\{1\}}, P_{\{2\}} \,\}$

$$\text{(Iteration 2):}\quad \mathsf{Cl}_{v_1,v_1} = \mathsf{Cl}_{v_1,v_2} = \mathsf{Cl}_{v_2,v_1} = \mathsf{Cl}_{v_2,v_2} = \varnothing \quad \mathsf{Cl}_{v_1,v_3} = \{\, P_{\{1\}} \,\} \quad \mathsf{Cl}_{v_2,v_3} = \{\, P_{\{2\}} \,\}$$
$$\mathsf{Cl}_{v_3,v_1} = \mathsf{Cl}_{v_3,v_2} = \{\, P_{\varnothing} \,\} \quad \mathsf{Cl}_{v_3,v_3} = \mathsf{Cl}_{v_3,v_3} = \{\, P_{\{1\}}, P_{\{2\}}, P_{\{3\}} \,\}$$
$$\text{(Iteration 3):}\quad \mathsf{Cl}_{v_1,v_1} = \mathsf{Cl}_{v_1,v_2} = \mathsf{Cl}_{v_1,v_3} = \{\, P_{\{1\}}, P_{\{1,2\}}, P_{\{1,3\}}, P_{\{1,2,3\}} \}$$
$$\mathsf{Cl}_{v_2,v_1} = \mathsf{Cl}_{v_2,v_2} = \mathsf{Cl}_{v_2,v_3} = \{\, P_{\{2\}}, P_{\{1,2\}}, P_{\{2,3\}}, P_{\{1,2,3\}} \}$$
$$\mathsf{Cl}_{v_3,v_1} = \mathsf{Cl}_{3,2} = \{\, P_{\varnothing}, P_{\{1\}}, P_{\{2\}}, P_{\{3\}}, P_{\{1,2\}}, P_{\{1,3\}}, P_{\{2,3\}}, P_{\{1,2,3\}} \}$$
$$\mathsf{Cl}_{v_3,v_3} = \{\, P_{\{1\}}, P_{\{2\}}, P_{\{3\}}, P_{\{1,2\}}, P_{\{1,3\}}, P_{\{2,3\}}, P_{\{1,2,3\}} \}$$

The minimal composition closure contains only the $<$-minimal sloped relations in each set, which we have highlighted in grey. Taking the ordering $v_1 < v_2 < v_3$ on the vertices, the order-reduced composition closure is as follows (again, with $\leq$-minimal relations highlighted in grey):

$$\Omega_{v_1,v_1} = \Omega_{v_1,v_2} = \Omega_{v_2,v_1} = \Omega_{v_2,v_2} = \varnothing \quad \Omega_{v_1,v_3} = \{\, P_{\{1\}} \,\} \quad \Omega_{v_2,v_3} = \{\, P_{\{2\}} \,\}$$
$$\Omega_{v_3,v_1} = \Omega_{v_3,v_2} = \{\, P_{\varnothing} \,\} \quad \Omega_{v_3,v_3} = \mathsf{Cl}_{v_3,v_3} = \{\, P_{\{1\}}, P_{\{2\}}, P_{\{3\}}, P_{\{1,2\}}, P_{\{1,3\}}, P_{\{2,3\}}, P_{\{1,2,3\}} \}$$

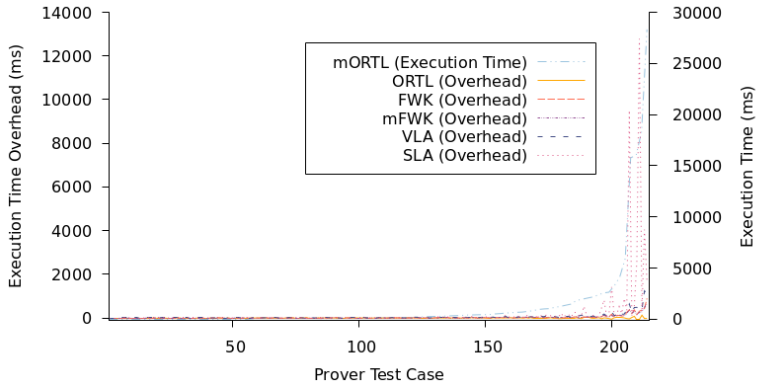
In general, we have that only $O(n)$ elements in the order-reduced closure are non-empty. In contrast, all n^2 elements of the full composition closure contain $O(2^w)$ relations.

Notice that the Transitive Looping algorithm builds up the composition closure iteratively: Each iteration considers *all* pairs of vertices, adding the sloped relations corresponding to paths travelling via the intermediate vertex being considered by the current iteration. In contrast, the Order-reduced Transitive Looping algorithm computes each closure element 'in one go', according to the fixed ordering on vertices, i.e. in the following order: $\Omega_{v_1,v_1}, \Omega_{v_1,v_2}, \Omega_{v_1,v_3}, \Omega_{v_2,v_1}, \ldots, \Omega_{v_3,v_3}$. What makes the algorithm faster is that, for each pair of vertices, it considers only sloped relations corresponding to paths travelling through intermediate nodes less than or equal to the end point. So, the computation of Ω_{v_1,v_1} does not need to consider any other closure element, computing only $\{\mathsf{R}_{v_1,v_1}\}^+$. The computation of Ω_{v_2,v_2} then uses only Ω_{v_2,v_1} and Ω_{v_1,v_2}, calculating $(\{\mathsf{R}_{v_2,v_2}\} \cup (\Omega_{v_2,v_1} \circ \Omega_{v_1,v_2}))^+$. Finally, Ω_{v_3,v_3} is computed as $(\{\mathsf{R}_{v_3,v_3}\} \cup (\Omega_{v_3,v_1} \circ \Omega_{v_1,v_3}) \cup (\Omega_{v_3,v_2} \circ \Omega_{v_2,v_3}))^+$. Thus algorithm 2 computes (sometimes) significantly fewer compositions of sloped relations than algorithm 1.

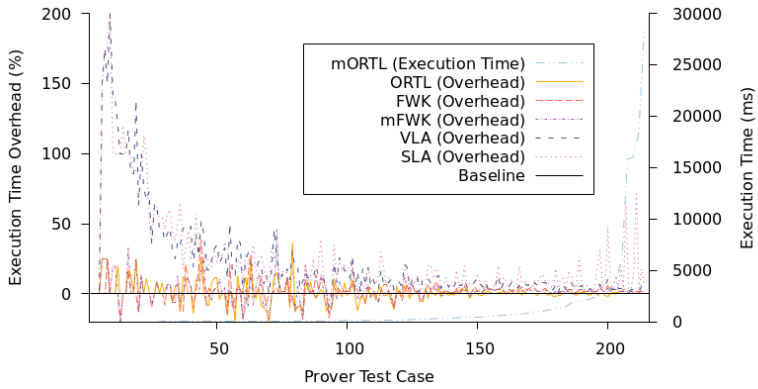
6 IMPLEMENTATION AND EVALUATION IN CYCLIST

We now describe a comparative analysis of the practical performance of the various criteria for deciding Infinite Descent, complementing our theoretical complexity analysis. Namely, we evaluate implementations of the Vertex-language Automaton criterion (VLA), the Slope-language Automaton criterion (SLA), and the full and minimal versions, respectively, of both the Transitive Looping criterion using the Floyd-Warshall-Kleene algorithm (FWK and mFWK) and the Order-reduced Transitive Looping criterion (ORTL and mORTL). The implementations were all carried out in C++ and integrated within the Cyclist prover framework [Brotherston et al. 2012]. Those for the automaton-based criteria use the API of the Spot model checker (v2.10.4) [Duret-Lutz et al. 2016] to create the path and trace automata, and then check inclusion. All experiments were carried out on an Intel(R) Xeon(R) CPU E5-2683 v4 operating at 2.1 GHz, running Ubuntu 20.04 LTS with 16 GB of RAM. Our implementation and evaluation data are available on Zenodo [Cohen et al. 2023], and our algorithms are incorporated into Cyclist's codebase [Cyclist Project Developers 2023].

Performance on Practical examples. We evaluated our implementations using Cyclist's test suite of 'real-world' Separation Logic and first-order logic examples, consisting of 233 logical entailments. Cyclist performs an iterative depth-first search procedure to find proofs, during which it makes many calls to the soundness check for the (partial) proofs it finds. We timed how long Cyclist took to find proofs using each method, averaging over 5 test runs. Cyclist finds proofs for 216 test cases within a 60 s timeout. 7 test cases took negligible time ($\leq$1 ms), and don't admit reasonable



(a) Comparison of Absolute Overhead

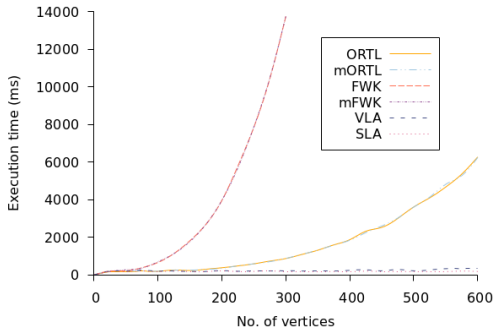


(b) Comparison of % Overhead

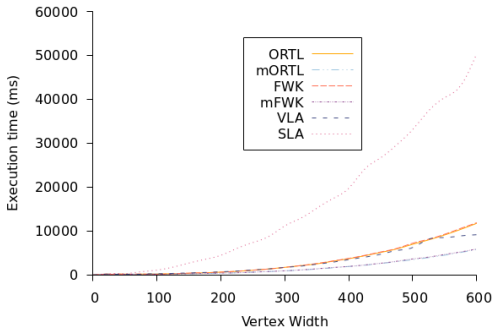
Fig. 6. Average-case Performance

comparison. In the other cases, none of ORTL, VLA and SLA ever had the fastest average execution time. mORTL had the fastest average time in 106 cases, and FWK and mFWK in 15 and 18 cases, respectively. ORTL, mORTL, FWK and mFWK had joint fastest average times with one or more of the other criteria in 41, 53, 41 and 35 cases, respectively. By comparing the overhead in average execution time of each method against the others we found that mORTL performs best overall, although the performance of ORTL, mORTL, FWK, and mFWK all seem to be broadly similar. We found that ORTL, FWK, and mFWK all had a mean overhead of 1%–2% compared with mORTL. The automaton-based methods performed comparably to each other, but were clearly worse than the other methods, with mean overheads of 22% and 25% for VLA and SLA, respectively. There were some notable outliers for SLA: We observed two test cases for which SLA took 12.7 s and 9.5 s longer than mORTL. Fig. 6 plots the overhead in both absolute and percentage execution time of each method compared against mORTL, on the left-hand y-axis. The test cases are ordered increasingly by baseline execution time, which is plotted on the right-hand y-axis for comparison.

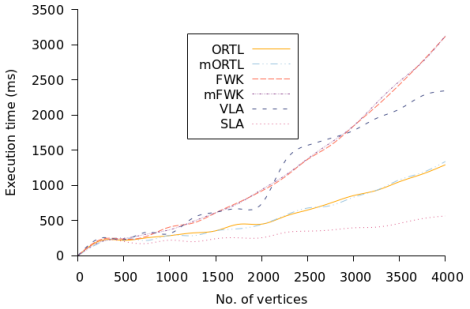
We also collected statistics about the number and complexity of the sloped graphs encountered and checked by Cyclist during these benchmark tests. The maximum number of Infinite Descent checks performed in any given test case was 2242, with a mean and median of 111 and 19, respectively. Over the whole benchmark suite, the maximum number of vertices encountered was 16, and



(a) Family A Benchmarks



(b) Family B Benchmarks



(c) Family C Benchmarks

Fig. 7. Comparison of Performance of Different Methods

the maximum vertex width was 14. The median number of vertices and vertex width were 3 and 6, respectively. This shows that the size of sloped graphs encountered in practice is quite small, but that a significant number of infinite descent checks are required during proof search. Moreover, it shows sloped graphs 'in the wild' tend to have a larger vertex width than number of vertices.

Worst-case Performance. We also compared all methods on three hand-crafted families of tests aimed at poor/worst-case performance.

Family A (Fig. 7a) includes sloped graphs with varying numbers of vertices, each having a single position and involved in a distinct cycle with a 'root' vertex. The positions are all connected and decrease on each cycle, so the graphs satisfy Infinite Descent. We tested on a range of sizes to demonstrate execution time profiles. The execution time of ORTL and mORTL display something like a polynomial profile (which is at most cubic by our theoretical complexity analysis) and hits 6.2 s with a graph of 600 vertices. The performance of both order-reduced implementations is the same here. This is because the vertex width is 1, so each of the closure sets contains at most a single sloped relation, and there is no scope for filtering out any sloped relations. FWK and mFWK also display the same curve as each other, for the same reason. This curve is also polynomial, but is much steeper than that for ORTL/mORTL. The Floyd-Warshall-Kleene algorithm performs particularly badly on this family because, although it manages to compute the entire closure after just two iterations of its outermost loop, it must perform the remaining iterations, which then compute the same closure set over and over. VLA and SLA show an almost linear profile for this class, reaching an execution time of 0.33 s and 0.17 s, respectively, for a graph of 600 vertices. Clearly, the relation-based approaches do not compete here. However, the Spot model checker (used to implement VLA and SLA) suddenly hangs on graphs with more than 4094 vertices—possibly

because its optimizations are running out of steam for larger values and an exponential profile is taking over (though the exact reason is not clear to us).

Family B (Fig. 7b) includes sloped graphs consisting of a single vertex and a varying number of positions connected in a chain, with only one pair of positions connected by a 'decrease' slope. These graphs also satisfy Infinite Descent. Again, ORTL and mORTL display polynomial curves, reaching execution times of 11.7 s and 5.8 s, respectively, for a vertex width of 600. For this family, since there is only a single vertex, FWK/mFWK and ORTL/mORTL are essentially the same algorithm: computing the composition closure of the singleton set containing the sloped relation along the edge from the single vertex to itself. The graph shows that, indeed, ORTL and FWK have the same curve as each other, as do mORTL and mFWK. Here, we also see the benefit of using the minimal order-reduced closure: The graph suggests that the size of the closure sets are roughly halved. We observe that VLA performs similarly to ORTL/FWK but that SLA performs significantly worse, reaching an execution time of 50 s for a graph with a vertex width of 600. This is rather unexpected given our complexity analysis, which results in better worst-case complexity for SLA when $n = 1$. We do not have a clear idea of why Spot would demonstrate this discrepancy. Moreover, for this family of graphs, we ran into a limitation of Spot concerning a compile-time parameter specifying the width of a bit vector used in its optimisations. Namely, the default value of this parameter only allowed Spot to deal with up to 16 positions, so we had to recompile it with a larger value for the parameter.

Family C (Fig. 7c) is a variation of Family A, in which only *every other* cycle allows a decrease in the position. Thus, these graphs do not satisfy Infinite Descent. For the same reason as above, we see ORTL/mORTL and FWK/mFWK displaying the same respective execution profiles. However, on this family, the fast-fail ability of the relation-based approaches came into their own, compared to the performance on Family A: ORTL and mORTL take only 1.3 s to check a graph with 4000 vertices; FWK and mFWK take 3.1 s. Our implementations of the order-reduced criteria outperform VLA, which takes 2.3 s to check a graph of 4000 vertices, but we see VLA beginning to outperform the standard relation-based algorithm. However, all of these methods are outperformed by SLA. The better performance of SLA compared to VLA aligns with our complexity analysis, which for $w = 1$ gives polynomal complexity for SLA versus exponential for VLA. However, it is not clear to us why the performance of the automaton-based criteria are so different on this family, whereas they are very similar on Family A. Further investigation into the optimisations employed by Spot might provide some insight. Although the relation-based methods are not the best performing on this family, their execution profiles grow smoothly up to a very large number of vertices, whilst VLA and SLA, using Spot, again hang on graphs with more than 4094 vertices.

Summary. Our implementation of the minimal order-reduced Transitive Looping criterion displays improvements over the Transitive Looping and automaton-based approaches in several worst-case and real-world contexts. Our worst-case benchmarks suggest that the mORTL criterion performs better than the VLA criterion on negative examples, and better than both automata-based approaches on positive examples with low number of vertices and high vertex width. Given that the sloped graphs encountered by Cyclist have a high vertex width to number of vertices ratio and that a high proportion of sloped graphs checked during any given proof search do not satisfy Infinite Descent, this may explain the favourable results for the order-reduced criteria on real-world examples.

7 RELATED WORK

Below we survey other studies on criteria for Infinite Descent and related properties.

The Size-Change Principle. Lee et al. already describe the Vertex-language Automaton criterion and Transitive Looping criterion as methods for deciding size-change termination. In this paper, we describe novel automata-theoretic and relation-based algorithms that improve on these. Some

experimental evaluation of the Transitive Looping criterion was carried out by Ben-Amram and Lee [2007], consisting of a sizeable test suite (123 cases), but the examples (and execution times) were very small so it does not demonstrate how the algorithm scales. Experimental evaluation of 5 small examples was also carried out by Fogarty and Vardi [2009]. The Size-Change Principle has been explicitly employed recently in the setting of functional programs [Jones et al. 2022; Lepigre and Raffalli 2019]. Evaluating our algorithms against these implementations is an interesting area for future work. The Transitive Looping criterion was also adapted for linear-time temporal logic [Lange 2011].

Slope-language Automata. Our slope-language automaton construction has an interesting precursor in the literature, which can be seen to already contain the germ of the idea of replacing vertices with sloped relations: Dax et al. [2006] consider an alternative automaton-based construction for checking provability within a cyclic system for the linear time μ-calculus. This still checks an inclusion between a path and trace automaton. However, the states of the path automaton consist not of vertices in a particular pre-proof (sequents in their case), but rather subsets of the Fischer-Ladner closure of a formula, F. In the setting of their proof system, any sequent occurring in a pre-proof deriving F can be identified with such a subset. We can interpret their system in our abstract framework by taking formulas to be abstract positions. The alphabet of their automata construction consists of the names of inference rules in the proof system, annotated with an eigenformula. It turns out that these annotated rule names actually contain all the information necessary to reconstruct the sloped relations that arise in the abstract setting, due to a particularly convenient coincidence: Along an edge (v, v'), each position in $\mathsf{Ps}(v') \setminus \mathsf{Ps}(v)$ derives from only one position in $\mathsf{Ps}(v) \setminus \mathsf{Ps}(v')$, namely the eigenformula in v. Moreover, each other position in $\mathsf{Ps}(v')$ is always and only ($\rightsquigarrow$)-related to its corresponding occurrence in v. Despite the similarity to our Slope-language Automata, there is still an important difference: The construction of Dax et al. [2006] checks for provability of a given formula, rather than Infinite Descent in a particular preproof. The states of the path automaton are sequents that may occur in *some* pre-proof of F, meaning that the automaton effectively represents all possible pre-proofs of F *simultaneously*. The automata inclusion check thus ensures that every possible pre-proof deriving F is actually a valid proof. This can be done for well-behaved logics (like the μ-calculus), where there exists a bound on the size of proofs of a given formula and the automata can essentially consider paths in all potential proofs of the formula at once. However, this is not possible in general. In contrast, our construction, which works for any cyclic proof system (captured by our abstract framework), uses the nodes of a given pre-proof as states in the path automaton and "only" checks whether that particular pre-proof satisfies infinite descent, rather than whether the derived formula/sequent has any proof at all.

Parity-based Trace Conditions. Our abstract formulation of Infinite Descent does not immediately encapsulate parity-based trace conditions from the literature, e.g., for μ-calculus [Baelde et al. 2016] or hypersequents [Das and Girlando 2022]. However, the automata-theoretic constructions we have described may be easily adapted to use parity automata rather than Büchi automata, and used to decide these trace conditions [Das and Girlando 2022; Dax et al. 2006]. In fact, it is not too difficult to see that our abstract framework itself can be generalised to capture these conditions. Instead of adopting the two-element total order $\{\rightsquigarrow < \not\searrow\}$ as the set of slopes, we can take any arbitrary (finite) total order, marking each alternating element as even and odd, respectively. A trace of such slopes is descending if the maximum slope that appears infinitely often is odd. The notion of descending trace in definition 2.13 is simply a special case of this. The relation-based algorithms using the full (order-reduced) composition closure can be similarly adapted, by requiring the transitive looping check to verify that a sloped relation (when viewed as a graph) has a strongly connected component whose maximum slope is odd. However, in this generalised context the approximation-based optimisation is no longer sound.

Incomplete Criteria. Our attention in this paper was on *complete* criteria, but the literature contains several criteria only sufficient (but not necessary) for verifying Infinite Descent. These include the criterion of Sprenger and Dam [2002, 2003] for (first-order) μ-calculus, as well as Brotherston's [2006, Ch. 7] trace manifold condition. The latter condition is more permissive than the former [Brotherston 2006, Example 7.2.4]. A different approach to the checking of Infinite Descent is taken in a series of papers by Stratulat, who introduces a sound but incomplete polynomial criterion based on ordering constraints that are checked along paths in the minimal cycles of the cyclic proof trees [Stratulat 2017, 2018]. This criterion is further improved by replacing the (potentially expensive) enumeration of minimal cycles by the computation of a normal form, and is demonstrated to perform better in practice than the standard automata-based check [Stratulat 2021]. On the other hand, Stratulat's approach is not described in general for any cyclic proof system, but is tailored to an (important) particular case: that of cyclic proof systems for FOLID—first-order logic with inductive definitions [Brotherston and Simpson 2011]. Indeed, it takes advantage of concrete details of the logic, including the presence of well-defined substitutions. However, it seems plausible that the results can be generalized to a class of logical systems having certain abstractly describable features. These incomplete criteria also seem related to 'reset' proof systems [Jungteerapanich 2010; Wehr 2023], which validate cycles using local, finitary conditions. In future work, we aim to analyse the incomplete criteria from the literature in our abstract framework, implement them and evaluate their performance and coverage compared to the complete methods. Combined with the methods already implemented, this could lead to a Sledgehammer-like tool [Paulson and Blanchette 2010] for checking Infinite Descent.

Tractable Restrictions. The PSPACE-completeness result shows that, in general, it is not possible to reduce the complexity in both parameters. However, our parameterised complexity analysis shows that the Slope-language Automaton and (Order-reduced) Transitive Looping algorithms are overall polynomial when the number of vertices is exponentially larger than the trace width, providing several examples justifying an observation made by Hazard and Kuperberg [2022, Remark 17]. Wehr [2023] also notes that checking Infinite Descent in 'reset' proof systems is polynomial, but that converting a general cyclic pre-proof into a reset pre-proof involves an exponential blowup in general. Ben-Amram and Lee [2007] described a polynomial time algorithm for deciding Size-Change Termination on a restricted range of inputs, demonstrating that this covers a large proportion of practical cases in program termination. It should be possible to describe this restriction on the level of our sloped graphs. We also speculate that alternative restrictions in terms of the sloped relations occurring in the input graph could result in the (Order-reduced) Transitive Looping algorithms already running in polynomial time. Such restrictions are orthogonal to the incomplete criteria considered above, which apply to all inputs but may return a "don't know" answer.

8 CONCLUSION

This paper studied criteria for checking Infinite Descent in the context of cyclic proof theory. We focused on the parameterized complexity of automaton-based and relation-based criteria and developed novel generalizations and optimizations. Our analysis has identified relative strengths and limitations. Additionally, we gave prototype implementations of our novel criteria and evaluated their practical performance, which could inform the development of more effective methods.

We plan to further explore the relationship between our order-reduced transitive looping criterion, graph search strategies, and general Büchi complementation strategies, following Fogarty and Vardi [2009, 2010]. Also, for certain classes of Büchi automata, the inclusion problem can be decided in polynomial time [Angluin and Fisman 2020; Bousquet and Löding 2010]. Translating the defining properties of these classes from automata to sloped graphs might lead to structural restrictions at the proof-system level ensuring tractability of Infinite Descent.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for insightful suggestions. Their observations have helped us improve the presentation, correct some errors, and use a more appropriate baseline algorithm for the relation-based criteria. We also thank the reviewers of a previous version of this paper for observations leading to a stronger focus on parameterized complexity, the development of the slope-language automaton criterion, and significant clarification of the order-reduced relation-based criterion. We gratefully acknowledge support from the UK Royal Society travel grant "Cyclic Reasoning Mechanisms for Interactive Theorem Proving".

DATA-AVAILABILITY STATEMENT

The artifact associated with this paper (comprising the implementation of our algorithms integrated with the Cyclist prover framework, and the experimental data) is available as a Zenodo deposit [Cohen et al. 2023]. Our implementations are also included within the public source code repository of the Cyclist prover framework, on Github (https://github.com/ngorogiannis/cyclist). The specific commit in the Cyclist Github repository that corresponds to our implementation at the time of publication of this article is labeled with the tag POPL2024 [Cyclist Project Developers 2023].

REFERENCES

Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukás Holík, Chih-Duo Hong, Richard Mayr, and Tomás Vojnar. 2010. Simulation Subsumption in Ramsey-Based Büchi Automata Universality and Inclusion Testing. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6174)*, Tayssir Touili, Byron Cook, and Paul B. Jackson (Eds.). Springer, 132–147. https://doi.org/10.1007/978-3-642-14295-6_14

Bahareh Afshari and Dominik Wehr. 2022. Abstract Cyclic Proofs. In *Logic, Language, Information, and Computation - 28th International Workshop, WoLLIC 2022, Iaşi, Romania, September 20-23, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13468)*, Agata Ciabattoni, Elaine Pimentel, and Ruy J. G. B. de Queiroz (Eds.). Springer, 309–325. https://doi.org/10.1007/978-3-031-15298-6_20

Dana Angluin and Dana Fisman. 2020. Polynomial Time Algorithms for Inclusion and Equivalence of Deterministic Omega Acceptors. *CoRR* abs/2002.03191 (2020). arXiv:2002.03191 https://arxiv.org/abs/2002.03191

David Baelde, Amina Doumane, and Alexis Saurin. 2016. Infinitary Proof Theory: the Multiplicative Additive Case. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France (LIPIcs, Vol. 62)*, Jean-Marc Talbot and Laurent Regnier (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 42:1–42:17. https://doi.org/10.4230/LIPIcs.CSL.2016.42

Amir M. Ben-Amram and Chin Soon Lee. 2007. Program Termination Analysis in Polynomial Time. *ACM Trans. Program. Lang. Syst.* 29, 1 (2007), 5:1–5:37. https://doi.org/10.1145/1180475.1180480

Stefano Berardi and Makoto Tatsuta. 2017. Classical System of Martin-Löf's Inductive Definitions Is Not Equivalent to Cyclic Proof System. In *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10203)*, Javier Esparza and Andrzej S. Murawski (Eds.). 301–317. https://doi.org/10.1007/978-3-662-54458-7_18

Stefano Berardi and Makoto Tatsuta. 2019. Classical System of Martin-Lof's Inductive Definitions is not Equivalent to Cyclic Proofs. *Log. Methods Comput. Sci.* 15, 3 (2019). https://doi.org/10.23638/LMCS-15(3:10)2019

Nicolas Bousquet and Christof Löding. 2010. Equivalence and Inclusion Problem for Strongly Unambiguous Büchi Automata. In *Language and Automata Theory and Applications, 4th International Conference, LATA 2010, Trier, Germany, May 24-28, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6031)*, Adrian-Horia Dediu, Henning Fernau, and Carlos Martín-Vide (Eds.). Springer, 118–129. https://doi.org/10.1007/978-3-642-13089-2_10

James Brotherston. 2006. *Sequent Calculus Proof Systems for Inductive Definitions.* Ph. D. Dissertation. University of Edinburgh. https://era.ed.ac.uk/handle/1842/1458

James Brotherston, Richard Bornat, and Cristiano Calcagno. 2008. Cyclic proofs of program termination in separation logic. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, George C. Necula and Philip Wadler (Eds.). ACM, 101–112. https://doi.org/10.1145/1328438.1328453

James Brotherston and Nikos Gorogiannis. 2014. Cyclic Abduction of Inductively Defined Safety and Termination Preconditions. In *Static Analysis - 21st International Symposium, SAS 2014, Munich, Germany, September 11-13, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8723)*, Markus Müller-Olm and Helmut Seidl (Eds.). Springer, 68–84. https://doi.org/10.1007/978-3-319-10936-7_5

James Brotherston, Nikos Gorogiannis, and Rasmus Lerchedahl Petersen. 2012. A Generic Cyclic Theorem Prover. In *Programming Languages and Systems - 10th Asian Symposium, APLAS 2012, Kyoto, Japan, December 11-13, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7705)*, Ranjit Jhala and Atsushi Igarashi (Eds.). Springer, 350–367. https://doi.org/10.1007/978-3-642-35182-2_25

James Brotherston and Alex Simpson. 2011. Sequent calculi for induction and infinite descent. *J. Log. Comput.* 21, 6 (2011), 1177–1216. https://doi.org/10.1093/LOGCOM/EXQ052

Khoo Siau Cheng, Chin Wei Ngan, Ta Quang Trung, Le Ton Chanh, Aishwarya Sivaraman, and Nguyen Thanh Toan. 2016. Songbird Prover. https://songbird-prover.github.io/

Liron Cohen, Adham Jabarin, Andrei Popescu, and Reuben Rowe. 2023. The Complex(ity) Landscape of Checking Infinite Descent: Source Code and Evaluation Data. https://doi.org/10.5281/zenodo.10073582

Liron Cohen and Reuben N. S. Rowe. 2020. Non-Well-Founded Proof Theory of Transitive Closure Logic. *ACM Trans. Comput. Logic* 21, 4, Article 31 (Aug. 2020), 31 pages. https://doi.org/10.1145/3404889

The Cyclist Project Developers. 2023. The Cyclist Framework and Provers. https://github.com/ngorogiannis/cyclist/releases/tag/POPL2024

Anupam Das. 2020. On The Logical Complexity of Cyclic Arithmetic. *Log. Methods Comput. Sci.* 16, 1 (2020). https://doi.org/10.23638/LMCS-16(1:1)2020

Anupam Das and Marianna Girlando. 2022. Cyclic Proofs, Hypersequents, and Transitive Closure Logic. In *Automated Reasoning*, Jasmin Blanchette, Laura Kovács, and Dirk Pattinson (Eds.). Springer International Publishing, Cham, 509–528. https://doi.org/10.1007/s10817-023-09675-1

Constantinos Daskalakis, Richard M. Karp, Elchanan Mossel, Samantha J. Riesenfeld, and Elad Verbin. 2011. Sorting and Selection in Posets. *SIAM J. Comput.* 40, 3 (2011), 597–622. https://doi.org/10.1137/070697720

Christian Dax, Martin Hofmann, and Martin Lange. 2006. A Proof System for the Linear Time μ-Calculus. In *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings (Lecture Notes in Computer Science, Vol. 4337)*, S. Arun-Kumar and Naveen Garg (Eds.). Springer, 273–284. https://doi.org/10.1007/11944836_26

Amina Doumane. 2017. Constructive Completeness for the Linear-time μ-calculus. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*. 1–12. https://doi.org/10.1109/LICS.2017.8005075

Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. 2016. Spot 2.0 — A Framework for LTL and ω-automata Manipulation. In *Proceedings of the 14th International Symposium on Automated Technology for Verification and Analysis (ATVA'16) (Lecture Notes in Computer Science, Vol. 9938)*. Springer, 122–129. https://doi.org/10.1007/978-3-319-46520-3_8

Seth Fogarty and Moshe Y. Vardi. 2009. Büchi Complementation and Size-Change Termination. In *Tools and Algorithms for the Construction and Analysis of Systems, 15th International Conference, TACAS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5505)*, Stefan Kowalewski and Anna Philippou (Eds.). Springer, 16–30. https://doi.org/10.1007/978-3-642-00768-2_2

Seth Fogarty and Moshe Y. Vardi. 2010. Efficient Büchi Universality Checking. In *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6015)*, Javier Esparza and Rupak Majumdar (Eds.). Springer, 205–220. https://doi.org/10.1007/978-3-642-12002-2_17

Gerhard Gentzen. 1935. Untersuchungen über das Logische Schließen. I. *Mathematische Zeitschrift* 39, 1 (1935), 176–210. https://doi.org/10.1007/BF01201353

Emile Hazard and Denis Kuperberg. 2022. Cyclic Proofs for Transfinite Expressions. In *30th EACSL Annual Conference on Computer Science Logic, CSL 2022, February 14-19, 2022, Göttingen, Germany (Virtual Conference) (LIPIcs, Vol. 216)*, Florin Manea and Alex Simpson (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 23:1–23:18. https://doi.org/10.4230/LIPIcs.CSL.2022.23

Shachar Itzhaky, Hila Peleg, Nadia Polikarpova, Reuben N. S. Rowe, and Ilya Sergey. 2021. Cyclic program synthesis. In *PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021*, Stephen N. Freund and Eran Yahav (Eds.). ACM, 944–959. https://doi.org/10.1145/3453483.3454087

Eddie Jones, C.-H. Luke Ong, and Steven Ramsay. 2022. CycleQ: An Efficient Basis for Cyclic Equational Reasoning. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (San Diego, CA, USA) *(PLDI 2022)*. Association for Computing Machinery, New York, NY, USA, 395–409. https:

//doi.org/10.1145/3519939.3523731

Natthapong Jungteerapanich. 2010. *Tableau Systems for the Modal μ-Calculus.* Ph. D. Dissertation. University of Edingburgh. http://hdl.handle.net/1842/4208

Bakhadyr Khoussainov and Anil Nerode. 2001. *Automata Theory and Its Applications.* Birkhauser Boston, Inc., USA.

Martin Lange. 2011. Size-Change Termination and Satisfiability for Linear-Time Temporal Logics. In *Frontiers of Combining Systems, 8th International Symposium, FroCoS 2011, Saarbrücken, Germany, October 5-7, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6989)*, Cesare Tinelli and Viorica Sofronie-Stokkermans (Eds.). Springer, 28–39. https://doi.org/10.1007/978-3-642-24364-6_3

Chin Soon Lee, Neil D. Jones, and Amir M. Ben-Amram. 2001. The Size-change Principle for Program Termination. In *Conference Record of POPL 2001: The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, Chris Hankin and Dave Schmidt (Eds.). ACM, 81–92. https://doi.org/10.1145/360204.360210

Daniel J. Lehmann. 1977. Algebraic Structures for Transitive Closure. *Theor. Comput. Sci.* 4, 1 (1977), 59–76. https://doi.org/10.1016/0304-3975(77)90056-1

Rodolphe Lepigre and Christophe Raffalli. 2019. Practical Subtyping for Curry-Style Languages. *ACM Trans. Program. Lang. Syst.* 41, 1 (2019), 5:1–5:58. https://doi.org/10.1145/3285955

Rémi Nollet, Alexis Saurin, and Christine Tasson. 2019. PSPACE-Completeness of a Thread Criterion for Circular Proofs in Linear Logic with Least and Greatest Fixed Points. In *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11714)*, Serenella Cerrito and Andrei Popescu (Eds.). Springer, 317–334. https://doi.org/10.1007/978-3-030-29026-9_18

Lawrence C. Paulson and Jasmin Christian Blanchette. 2010. Three years of experience with Sledgehammer, a Practical Link Between Automatic and Interactive Theorem Provers. In *The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011 (EPiC Series in Computing, Vol. 2)*, Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska (Eds.). EasyChair, 1–11. https://doi.org/10.29007/36dt

Reuben N. S. Rowe and James Brotherston. 2017. Automatic cyclic termination proofs for recursive procedures in separation logic. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, Yves Bertot and Viktor Vafeiadis (Eds.). ACM, 53–65. https://doi.org/10.1145/3018610.3018623

Luigi Santocanale. 2002. A Calculus of Circular Proofs and Its Categorical Semantics. In *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2303)*, Mogens Nielsen and Uffe Engberg (Eds.). Springer, 357–371. https://doi.org/10.1007/3-540-45931-6_25

Sven Schewe. 2009. Büchi Complementation Made Tight. In *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings (LIPIcs, Vol. 3)*, Susanne Albers and Jean-Yves Marion (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, 661–672. https://doi.org/10.4230/LIPIcs.STACS.2009.1854

Cristina Serban and Radu Iosif. 2018. An Entailment Checker for Separation Logic with Inductive Definitions. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 76 (2018). https://doi.org/10.14279/TUJ.ECEASST.76.1073

Alex Simpson. 2017. Cyclic Arithmetic Is Equivalent to Peano Arithmetic. In *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10203)*, Javier Esparza and Andrzej S. Murawski (Eds.). 283–300. https://doi.org/10.1007/978-3-662-54458-7_17

Christoph Sprenger and Mads Dam. 2002. A note on global induction in a mu-calculus with explicit approximations. In *Fixed Points in Computer Science, FICS 2002, Copenhagen, Denmark, 20-21 July 2002, Preliminary Proceedings (BRICS Notes Series, Vol. NS-02-2)*, Zoltán Ésik and Anna Ingólfsdóttir (Eds.). University of Aarhus, 22–24. https://www.brics.dk/NS/02/2

Christoph Sprenger and Mads Dam. 2003. On the Structure of Inductive Reasoning: Circular and Tree-Shaped Proofs in the μ-Calculus. In *Foundations of Software Science and Computational Structures, 6th International Conference, FOSSACS 2003 Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2620)*, Andrew D. Gordon (Ed.). Springer, 425–440. https://doi.org/10.1007/3-540-36576-1_27

Sorin Stratulat. 2017. Cyclic Proofs with Ordering Constraints. In *Automated Reasoning with Analytic Tableaux and Related Methods - 26th International Conference, TABLEAUX 2017, Brasília, Brazil, September 25-28, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10501)*, Renate A. Schmidt and Cláudia Nalon (Eds.). Springer, 311–327. https://doi.org/10.1007/978-3-319-66902-1_19

Sorin Stratulat. 2018. Validating Back-links of FOLID Cyclic Pre-proofs. In *Proceedings Seventh International Workshop on Classical Logic and Computation, CL&C 2018, Oxford (UK), 7th of July 2018 (EPTCS, Vol. 281)*, Stefano Berardi and Alexandre Miquel (Eds.). 39–53. https://doi.org/10.4204/EPTCS.281.4

Sorin Stratulat. 2021. E-Cyclist: Implementation of an Efficient Validation of FOLID Cyclic Induction Reasoning. In *Proceedings of the 9th International Symposium on Symbolic Computation in Software Science, SCSS 2021, Hagenberg, Austria, September 8-10, 2021 (EPTCS, Vol. 342)*, Temur Kutsia (Ed.). 129–135. https://doi.org/10.4204/EPTCS.342.11

Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. 2016. Automated Mutual Explicit Induction Proof in Separation Logic. In *FM 2016: Formal Methods - 21st International Symposium, Limassol, Cyprus, November 9-11, 2016, Proceedings (Lecture Notes in Computer Science, Vol. 9995)*, John S. Fitzgerald, Constance L. Heitmeyer, Stefania Gnesi, and Anna Philippou (Eds.). 659–676. https://doi.org/10.1007/978-3-319-48989-6_40

Quang-Trung Ta, Ton Chanh Le, Siau-Cheng Khoo, and Wei-Ngan Chin. 2018. Automated lemma synthesis in symbolic-heap separation logic. *Proc. ACM Program. Lang.* 2, POPL (2018), 9:1–9:29. https://doi.org/10.1145/3158097

Robert Endre Tarjan. 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.* 1, 2 (1972), 146–160. https://doi.org/10.1137/0201010

Gadi Tellez and James Brotherston. 2020. Automatically Verifying Temporal Properties of Pointer Programs with Cyclic Proof. *J. Autom. Reason.* 64, 3 (2020), 555–578. https://doi.org/10.1007/s10817-019-09532-0

Dominik Wehr. 2023. *Representation Matters in Cyclic Proof Theory.* Licentiate Thesis. University of Gothenburg. https://hdl.handle.net/2077/75984

Mihalis Yannakakis. 1990. Graph-Theoretic Methods in Database Theory. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA*, Daniel J. Rosenkrantz and Yehoshua Sagiv (Eds.). ACM Press, 230–242. https://doi.org/10.1145/298514.298576

Received 2023-07-11; accepted 2023-11-07